US006545981B1

(12) **United States Patent**
Garcia et al.

(10) **Patent No.: US 6,545,981 B1**
(45) **Date of Patent: Apr. 8, 2003**

(54) **SYSTEM AND METHOD FOR IMPLEMENTING ERROR DETECTION AND RECOVERY IN A SYSTEM AREA NETWORK**

(75) Inventors: **David J. Garcia**, Los Gatos; **Richard O. Larson**, Cupertino, both of CA (US); **Stephen G. Low**; **William J. Watson**, both of Austin, TX (US)

(73) Assignee: **Compaq Computer Corporation**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/224,115**

(22) Filed: **Dec. 30, 1998**

Related U.S. Application Data

(60) Provisional application No. 60/070,650, filed on Jan. 7, 1998.

(51) **Int. Cl.**[7] **H04L 12/56**

(52) **U.S. Cl.** **370/242**; 370/248; 370/392; 714/48

(58) **Field of Search** 370/216, 217, 370/225, 252, 228–230, 242, 248, 257, 389, 394, 400, 465, 392; 340/2.1, 2.7, 3.43; 709/107, 239, 250, 238; 714/100, 48, 21; 379/221.01, 221.04

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,733,350	A		3/1988	Tone et al. 364/200
5,193,149	A	*	3/1993	Awiszio et al. 395/200
5,386,524	A		1/1995	Lary et al. 395/400
5,555,405	A		9/1996	Griesmer et al. 395/600
5,563,879	A		10/1996	Sanders et al. 370/60
5,574,849	A		11/1996	Sonnier et al. 395/182.1
5,619,274	A		4/1997	Roop et al. 348/461
5,675,579	A	*	10/1997	Watson et al. 370/248
5,678,007	A		10/1997	Hurvig 395/200.12
5,706,281	A	*	1/1998	Hashimoto et al. 370/352
5,737,595	A		4/1998	Cohen et al. 395/61
5,802,050	A		9/1998	Petersen et al. 370/394
5,920,886	A		7/1999	Feldmeier 711/108
5,991,797	A		11/1999	Furtral et al. 709/216
6,119,244	A		9/2000	Schoenthal et al. 714/4
6,272,591	B2		8/2001	Grun 711/114
6,347,337	B1		2/2002	Shah et al. 709/224
2002/0029305	A1		3/2002	Satran et al. 710/33

FOREIGN PATENT DOCUMENTS

EP	0 757 318 A2	2/1997	..... G06F/13/12

OTHER PUBLICATIONS

Garcia et al., "ServerNet II" *Parallel Computer Routing and Communication*, (2nd Int. WKSP), Jun. 26, 1997, pp. 119–135, XP002103164, Atlanta, GA.

Eicken Von T. et al., "U-Net: A User-Level Network Interrface for Parallel and Distributed Computing," *Operating Systems Review* (SIGOPS), vol. 29, No. 5, Dec. 1, 1995, pp. 40–53.

Dunning D. et al., "The Virtual Interface Architecture," *IEEE Micro*, vol. 18, No. 2, 3/98, pp. 66–76.

* cited by examiner

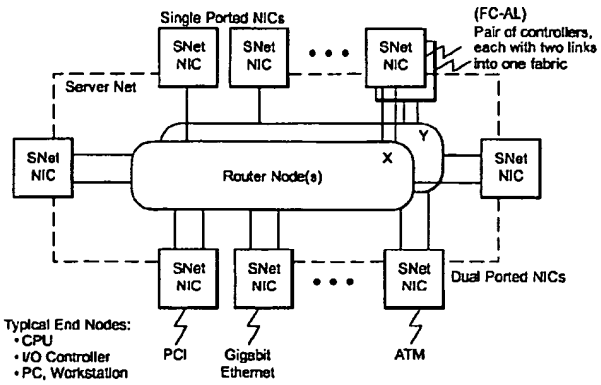
Primary Examiner—Huy D. Vu
Assistant Examiner—Duc Ho
(74) *Attorney, Agent, or Firm*—Oppenheimer, Wolff & Donnelly, LLP; Leah Sherry

(57) **ABSTRACT**

A system and method for facilitating both in-order and out-of-order packet reception in a SAN includes requestor and responder nodes, coupled by a plurality of paths, that maintain the good and bad status of each path and also maintain local copies of a message sequence number. If an error occurs for a transaction over a given path, the requestor informs the responder, over a good path, that the given path has failed and both nodes update their path status to indicate that the given path is bad. A barrier transaction is used by the requestor to determine whether the error is transient or permanent, and, if the error is transient, the requestor retries the transaction.

22 Claims, 6 Drawing Sheets

Fault Tolerant ServerNet II System Area Network



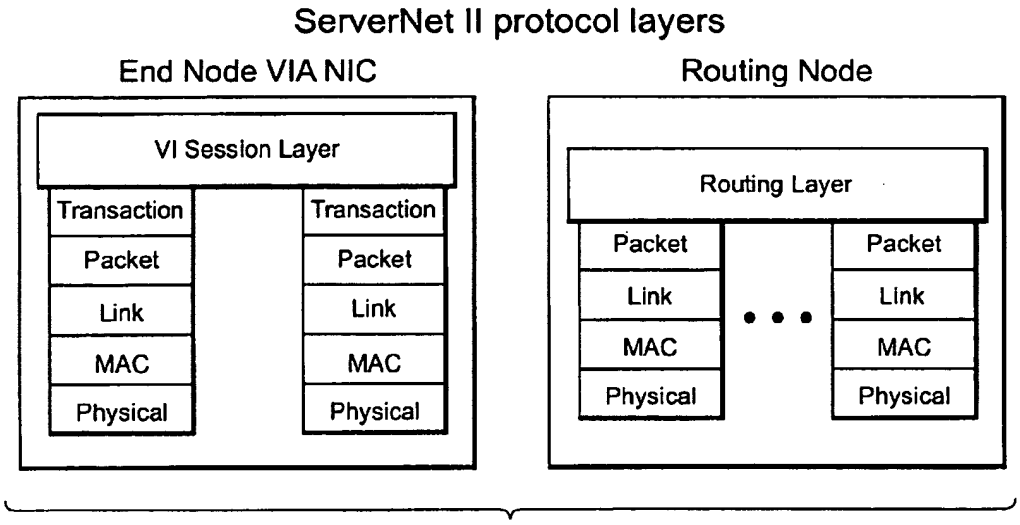

ServerNet II protocol layers
End Node VIA NIC
Routing Node
VI Session Layer
Transaction
Packet
Link
MAC
Physical
Transaction
Packet
Link
MAC
Physical
Routing Layer
Packet
Link
MAC
Physical
Packet
Link
MAC
Physical

FIG. 1

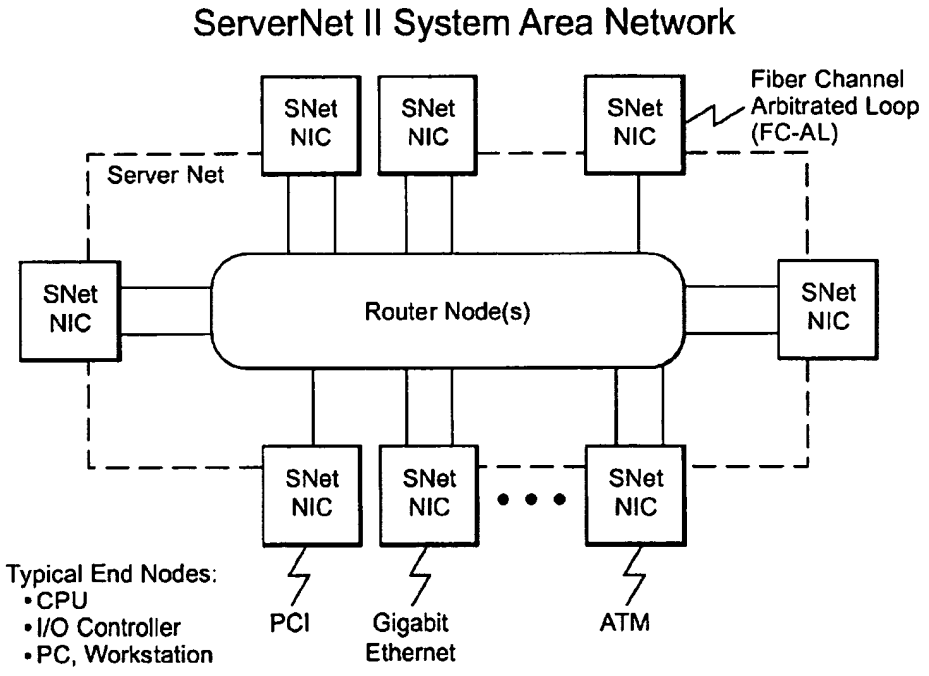

ServerNet II System Area Network
SNet NIC
SNet NIC
SNet NIC
Fiber Channel Arbitrated Loop (FC-AL)
Server Net
SNet NIC
Router Node(s)
SNet NIC
SNet NIC
SNet NIC
SNet NIC
Typical End Nodes:
•CPU
•I/O Controller
•PC, Workstation
PCI
Gigabit Ethernet
ATM

FIG. 2

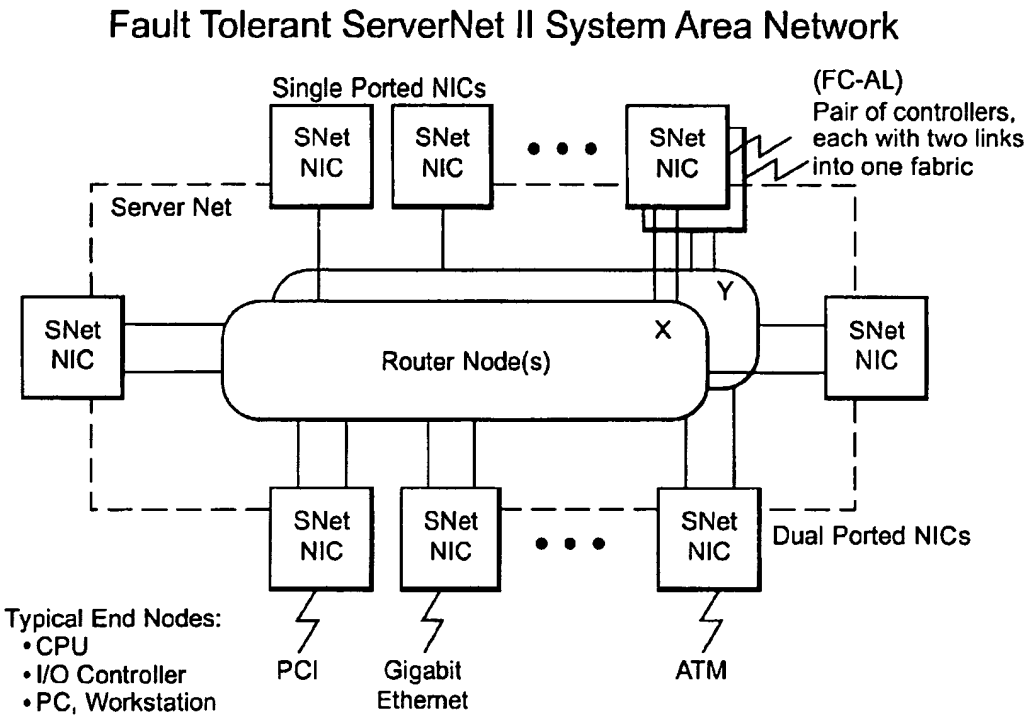

Fault Tolerant ServerNet II System Area Network
Single Ported NICs
(FC-AL)
Pair of controllers, each with two links into one fabric
SNet NIC
SNet NIC
SNet NIC
Server Net
Y
X
Router Node(s)
SNet NIC
SNet NIC
SNet NIC
SNet NIC
SNet NIC
Dual Ported NICs
Typical End Nodes:
• CPU
• I/O Controller
• PC, Workstation
PCI
Gigabit Ethernet
ATM

FIG. 3

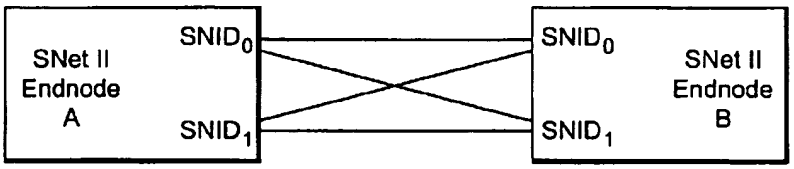

SNet II Endnode A
SNID0
SNID1
SNID0
SNID1
SNet II Endnode B
Four Logical Paths between SNet II Endnodes

FIG. 4

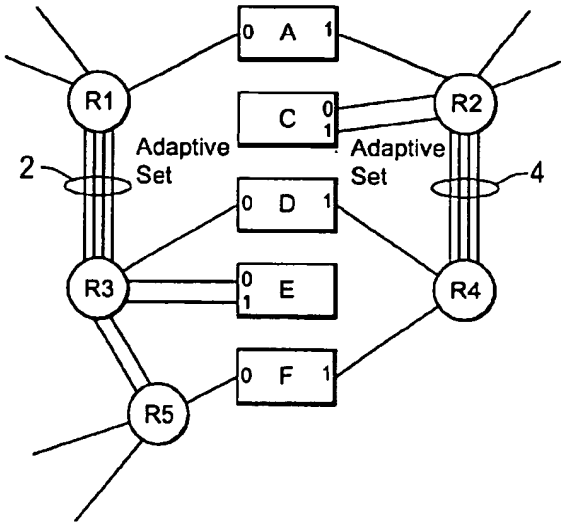


FIG. 5

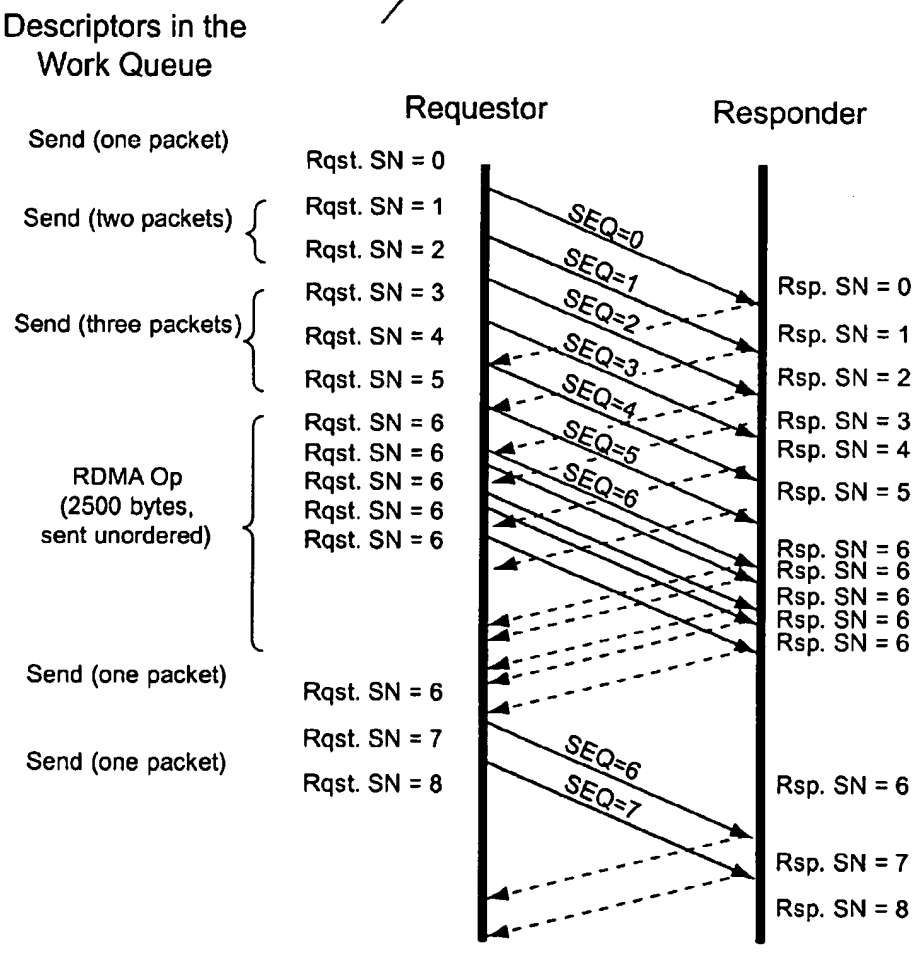


FIG. 6 Sequence Number Example

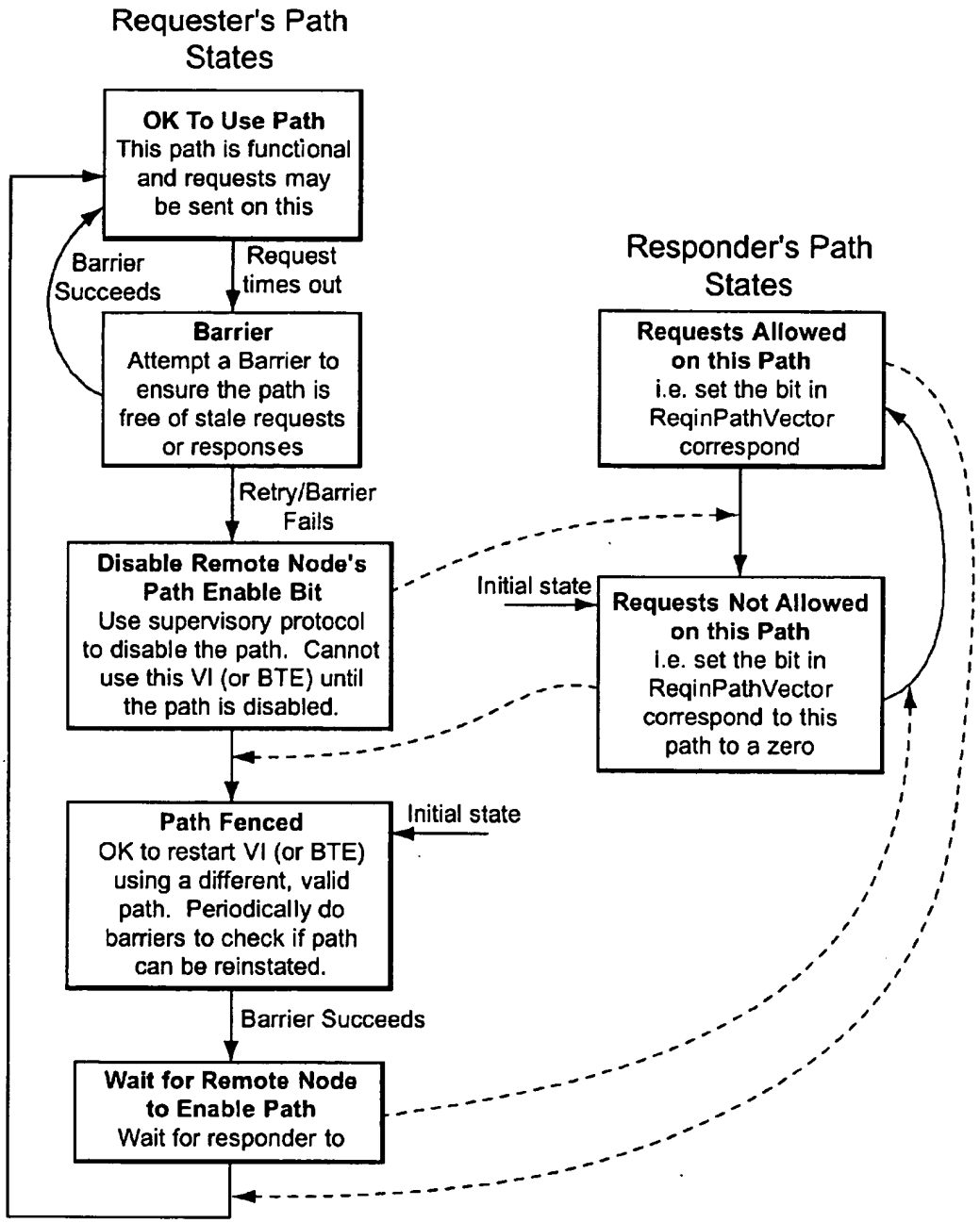
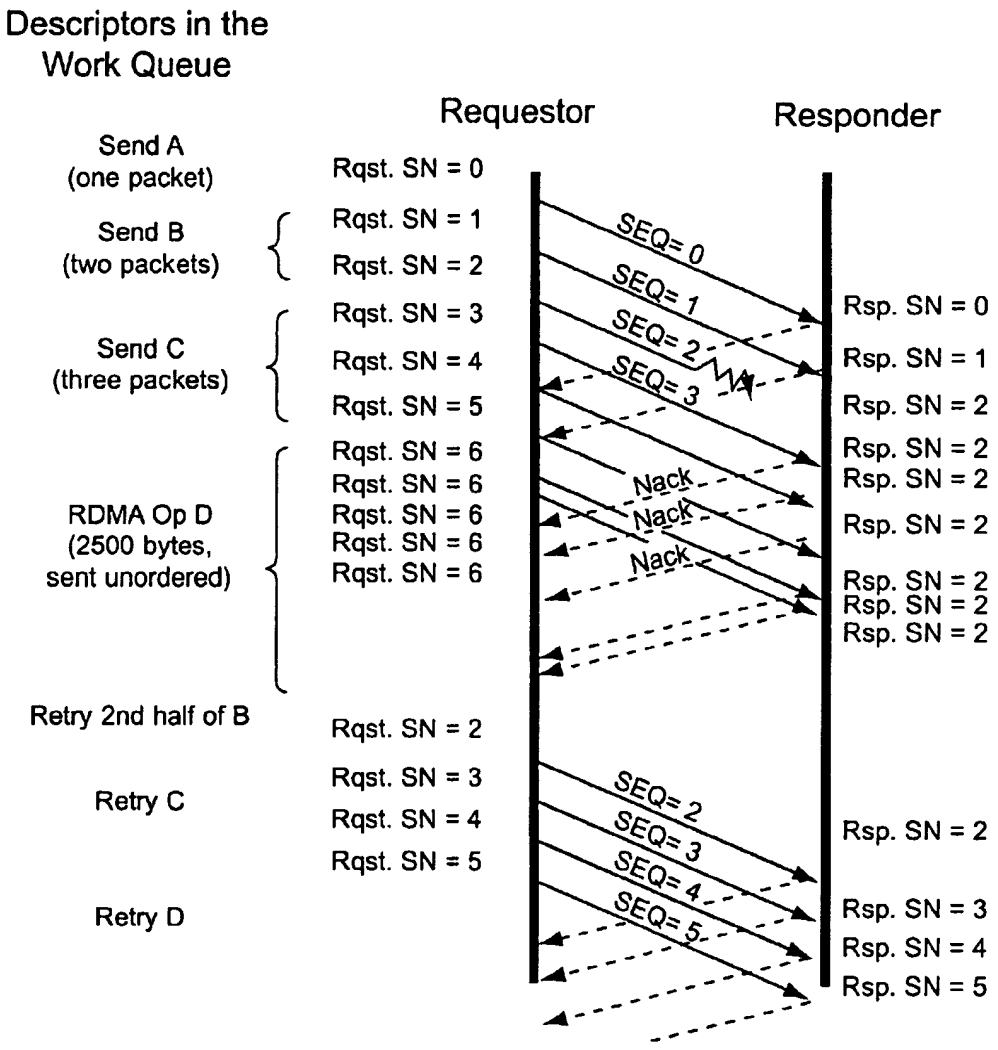
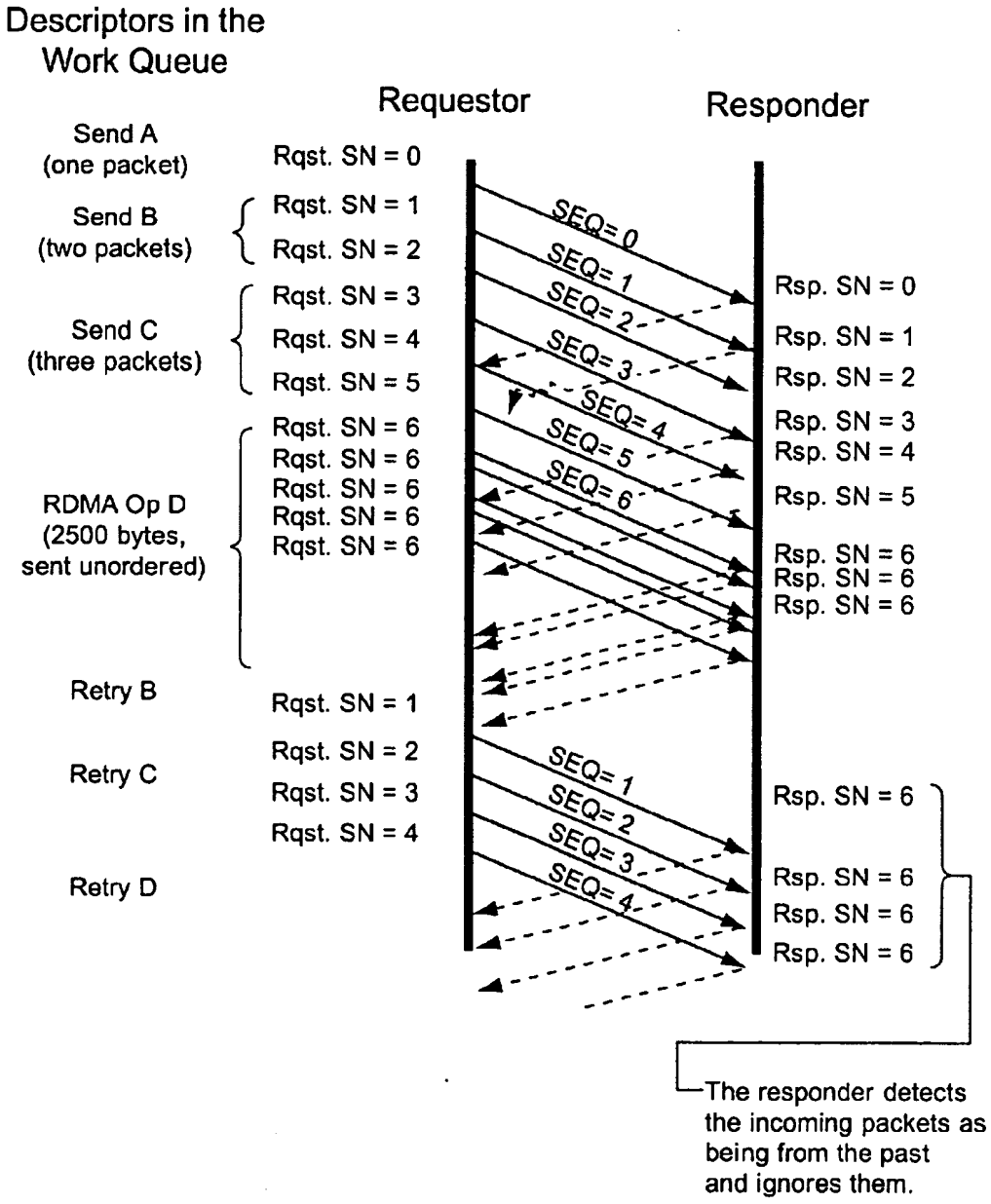


FIG. 7
Path State Diagram



Error Recovery - Lost Request

FIG. 8



Error Recovery - Lost Acknowledgment

FIG. 9

SYSTEM AND METHOD FOR IMPLEMENTING ERROR DETECTION AND RECOVERY IN A SYSTEM AREA NETWORK

CROSS-REFERENCES TO RELATED APPLICATIONS

This application claims priority from Provisional App. No. 60/070,650, filed Jan. 7, 1998, which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

Traditional network systems utilize either channel semantics (send/receive) or memory semantics (DMA) model. Channel semantics tend to be used in I/O environments and memory semantics tend to be used in processor environments.

In a channel semantics model, the sender does not know where data is to be stored, it just puts the data on the channel. On the sending side, the sending process specifies the memory regions that contain the data to be sent. On the receiving side, the receiving process specifies the memory regions where the data will be stored.

In the memory semantics model, the sender directs the data to a particular location in the memory, utilizing remote direct memory access (RDMA) transactions. The initiator of the data transfer specifies both the source buffer and destination buffer of the data transfer. There are two types of RDMA operations, read and write.

The virtual interface architecture (VIA) has been jointly developed by a number of computer and software companies. VIA provides consumer processes with a protected, directly accessible interface to network hardware, termed a virtual interface. VIA is especially designed to provide low latency message communication over a system area network (SAN) to facilitate multi-processing utilizing clusters of processors.

A SAN is used to interconnect nodes within a distributed computer system, such as a cluster. The SAN is a type of network that provides high bandwidth, low latency communication with a very low error rate. SANs often utilize fault-tolerant capability.

It is important for the SAN to provide high reliability and high-bandwidth, low latency communication to fulfill the goals of the VIA. Further, it is important for the SAN to be able to recover from errors and continue to operate in the event of equipment failures. Error recovery must be accomplished without high CPU overhead associated with all transactions. Furthermore, error recovery should not increase the complexity for the consumer of VIA services.

SUMMARY OF THE INVENTION

According to one aspect of the present invention, a SAN maintains local copies of a sequence number for each data transfer transaction at the requester and responder nodes. Each data transfer is implemented by the SAN as a sequence of request/response packet pairs. An error condition arises if a response to any request packet is not received at the requesting node. The responder and requestor nodes are coupled by a plurality of paths and each node maintains a record of the good or bad status of each path. If a transaction fails and the path is permanently bad, both nodes update their status to indicate that the path is bad. This is to prevent further transactions from including any stale requests that are potentially still in the network from arriving at the destination and potentially corrupting data.

According to another aspect of the invention, if an error occurs on a path the requestor node implements a barrier transaction on the path to determine if the failure is permanent or transient.

According to another aspect of the invention, the barrier transaction is performed by writing a number chosen from a large number space in a way that minimizes the probability of reusing the number in a short period of time.

According to one aspect of the invention, the number is randomly chosen from a large number space.

According to another aspect of the invention, the large number is based on the requestor ID and an incrementing component managed by the requestor.

According to another aspect of the invention, if the failure is transient the requestor retransmits packets starting with the packet that first caused an error condition to be detected.

According to another aspect of the invention, a sequence number is included in each request packet and copied into each response packet. A local copy of the sequence number is maintained at the requestor and responder nodes. If the sequence number in the request packet does not match the sequence number at the responder, a negative acknowledge response packet is generated.

Other features and advantages of the invention will be apparent in view of the following detailed description and appended drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram depicting ServerNet protocol layers implemented by hardware, where ServerNet is a SAN manufactured by the assignee of the present invention;

FIGS. **2** and **3** are block diagrams depicting SAN topologies;

FIG. **4** is a schematic diagram depicting logical paths between end nodes of a SAN;

FIG. **5** is a schematic diagram depicting routers and links connecting SAN end nodes;

FIG. **6** is a graph depicting the transmission of request and response packets between a requestor and a responder end node. FIG. **6** shows the sequence numbers used in packets for three Send operations, an RDMA operation, and two additional Send operations. The diagram shows the sequence numbers maintained in the requestor logic, the sequence number contained in each packet, and the sequence numbers maintained at the responder logic;

FIG. **7** is two interlocked state diagrams showing the state that software on the requestor and responder moves through for each path;

FIG. **8** is a graph depicting retransmission during error recovery due to a lost request packet; and

FIG. **9** is a graph depicting retransmission during error recovery due to a lost acknowledgment packet.

DESCRIPTION OF THE EMBODIMENTS

The preferred embodiments will be described implemented in the ServerNet II (ServerNet) architecture, manufactured by the assignee of the present invention, which is a layered transport protocol for a System Area Network (SAN) optimized to support the Virtual Interface (VI) architecture session layer which has stringent user-space to user-space latency and bandwidth requirements. These requirements mandate a reliable hardware (HW) message transport solution with minimal software (SW) protocol stack overhead. The ServerNet II protocol layers for an end

node VI Network Interface controller/Card (NIC) and for a routing node are illustrated in FIG. 1. A single NIC and VI session layer may support one or two ports, each with its associated transaction, packet, link-level, MAC (media access) and physical layer. Similarly, routing nodes with a common routing layer may support multiple ports, each with its associated link-level, MAC and physical layer.

Support for two ports enables the ServerNet II SAN to be configured in both non-redundant and redundant (fault tolerant, or FT) SAN configurations as illustrated in FIG. 2 and FIG. 3. On a fault tolerant network, a port of each end node may be connected to each network to provide continued VI message communication in the event of failure of one of the SANs. In the fault tolerant SAN, nodes may be ported into a single fabric or single ported end nodes may be grouped into pairs to provide duplex FT controllers. The fabric is the collection of routers, switches, connectors, and cables that connect the nodes in a network.

The following describes general ServerNet II terminology and concepts. The use of the term "layer" in the following description is intended to describe functionality and does not imply gate level partitioning.

Two ports are supported on a NIC for both performance and fault tolerance reasons. Both of these ports operate under the same session layer VI engine. That is, data may arrive on any port and be destined for any VI. Similarly, the VIs on the end node can generate data for any of these ports.

ServerNet II packets are comprised of a series of data symbols followed by a packet framing command. Other commands, used for flow control, virtual channel support, and other link management functions, may be embedded within a packet. Each request or response packet defines a variety of information for routing, transaction type, verification, length and VI specific information.

i. Routing in the ServerNet II SAN is destination-based using the first 3 bytes of the packet. Each NIC end node port in the network is uniquely defined by a 20 bit Port SNID (ServerNet Node ID). The first 3 bytes of a packet contain the Destination port's SNID or DID (destination port ID) field, a three bit Adaptive Control Bits (ACB) field and the fabric ID bit. The ACB is used to specify the path (deterministic or link-set adaptive) used to route the packet to its destination port as described in the following section.

ii. The transaction type fields define the type of session layer operation that this ServerNet II packet is carrying and other information such as whether it is a request or a response and, if a response, whether it is an ACK (acknowledgment) or a NACK (negative acknowledgment). The ServerNet II SAN also supports other transaction types.

iii. Transaction verification fields include the source port ID (SID) and a Transaction Serial Number. The transaction serial number enables a port with multiple requests outstanding to uniquely match responses to requests.

iv. The Length field consists of an encoding of the number of bytes of payload data in the packet. Payloads up to 512 bytes are supported and code space is reserved for future increases in payload size.

v. The VI Session Layer specific fields describe VI information such as the VI Operation, the VIA Sequence number and the Virtual Interface ID number. The VI Operation field defines the type of VI transaction being sent (Send, RDMA Read, RDMA Write) and other control information such as whether the packet is ordered or unordered, whether there is immediate data and/or whether this is the first or last packet in a session layer multi-packet transfer. Based on the VI transaction type and control information, a 32 bit Immediate data field or a 64 bit Virtual address may follow the VI ID number.

vi. The payload data field carries up to 512 bytes of data between requesters and responders and may contain a pad byte.

vii. The CRC field contains a checksum computed over the entire packet.

Transaction Overview

The basic flow of transactions through the ServerNet II SAN will now be described. VI requires the support of Send, RDMA read and RDMA write transactions. These are translated by the VI session layer into a set of ServerNet II transactions (request/response packet pairs). All data transfers (e.g., reading a disk file to CPU memory, dumping large volumes of data from a disk farm directly over a high-speed communications link, one end node simply interrupting another) consist of one or more such transactions.

Creating a Request Packet

The VI User Agent provides the low level routines for VIA Send, RDMA Write, and RDMA Read operations. These routines place a descriptor for the desired transfer in the appropriate VI queue and notify the VIA hardware that the descriptor is ready for processing. The VIA hardware reads the descriptor, and based on the descriptor contents, builds the ServerNet request packet header and assembles the data payload (if appropriate).

Dual Ports and Ordering

In a NIC with two ports, it is possible for a single VIA interface to process Sends and RDMA operations from several different VIs in parallel. It is also possible for a large RDMA transfer from a single VI to be transferred on both of the ports simultaneously. This latter feature is called Multi-pathing.

ServerNet II end nodes can connect both their ports to a single network fabric so that there are up to four possible paths between ServerNet II end nodes. Each port of a single end node may have a unique ServerNet ID (SNID). FIG. 4 depicts the four possible paths that End node A can use when sending request to End node B:

1) End node A SNID[0] to End node B SNID[0]
2) End node A SNID[0] to End node B SNID[1]
3) End node A SNID[1] to End node B SNID[0]
4) End node A SNID[1] to End node B SNID[1]

FIG. 5 depicts a network topology utilizing routers and links. In FIG. 5, end nodes A–F, each having first and second send receive ports 0 and 1, are coupled by a ServerNet topology including routers R1–R5. Links are represented by lines coupling ports to routers or routers to routers. A first adaptive set (fat pipe) 2 couples routers RI and R3 and a second adaptive set (fat pipe) 4 couples routers R2 and R4.

Routing may be deterministic or link set adaptive. An adaptive link-set is a set of links (also called lanes) between two routers that have been grouped to provide higher bandwidth. The Adaptive Control Bits (ACB) specify which type of routing is in effect for a particular packet.

Deterministic routing preserves strict ordering for packets sent from a particular source port to a destination port. In deterministic routing, the ACB field selects a single path or

lane through an adaptive link-set. Send transactions for a particular VI require strict ordering and therefore use deterministic routing.

RDMA transactions, on the other hand, may make use of all possible paths in the network without regard for the ordering of packets within the transaction. These transactions may use link-set adaptive routing as described below. The ACB field specifies which specific link (or lane) in this link-set is to be used for deterministic routing.

Alternatively, the ACB field can specify link-set adaptivity which enables the packets to dynamically choose from any of the links in the link-set.

A sample topology with several different examples of multipathing using link and path adaptivity is shown in FIG. 5.

Multipathing allows large block transfers done with RDMA Read or Write operations to simultaneously use both ports as well as adaptive links between the two communicating NICs. Since the data transfer characteristics of any one VI are expected to be bursty, multipathing allows the end node to marshal all its resources for a single transfer. Note that multipathing does not increase the throughput of multiple Send operations from one VI. Sends from one VI must be sent strictly ordered. Since there are no ordering guarantees between packets originating from different ports on a NIC, only one port may be used per Send. Furthermore, only a single ordered path through the Network may be used, as described in the following.

Transaction and Packet Layers

The transaction layer builds the ServerNet II request packet by filling in the appropriate SID, Transaction Serial Number (TSN), and CRC. The SID assigned to a packet always corresponds to the SNID of the port the packet originates from. The TSN can be used to help the port manage multiple outstanding requests and match the resulting responses uniquely to the appropriate request. The CRC enables the data integrity of the packet to be checked at the end node and by routers enroute.

Following the ServerNet II link protocol, the packet is encoded in a series of data symbols followed by a status command. The ServerNet II link layer uses other commands for flow control and link management. These commands may be inserted anywhere in the link data stream, including between consecutive data symbols of a packet. Finally, the symbols are passed through the MAC layer for transmission on the physical media to an intermediate routing node.

Routing

The routing control function is programmable so that the packet routing can be changed as needed when the network configuration changes (e.g., route to new end nodes). Router nodes serve as crossbar switches; a packet on any incoming (receive) side of a link can be switched to the outgoing (transmit) side of any link. As the incoming request packet arrives at a router node, the first three bytes, containing the DID and ACB fields, are decoded and used to select a link leading to the destination node. If the transmit side of the selected link is not busy, the head of the packet is sent to the destination node whether or not the tail of the packet has arrived at the routing node. If the selected link is busy with another packet, the newly arrived packet must wait for the target port to become free before it can pass through the crossbar.

As the tail of the packet arrives, the router node checks the packet CRC and updates the packet status (good or bad). The packet status is carried by a link symbol TPG (this packet good) or TPB (this packet bad) appended at the end of the packet. Since packet status is checked on each link, a packet status transition (good to bad) can be attributed to a specific link. The packet routing process described above is repeated for each router node in the selected path to the destination node.

Receiving a Request Packet

When the request packet arrives at the destination node, the ServerNet II interface receiver checks its validity (e.g., must contain correct destination node ID, the length is correct, the Fabric bit in the packet matches the Fabric bit associated with the receiving port, the request field encodes a valid request, and CRC must be good,). If the packet is invalid for any reason, the packet is discarded. The ServerNet II interface may save error status for evaluation by software. If these validity checks succeed, several more checks are made. Specifically, if the request specifies an RDMA Read or Write, the address is checked to ensure access has been enabled for that particular VI. Also, the input port and Source ID of the packet are checked to ensure access to the particular VI is allowed on that input port from the particular Source. If the packet is valid, the request can be completed.

Response Packet

A response is created based on the success (ACK response) or failure (NACK response) of the request packet. A successful read request, for example, would include the read data in the ACK response. The source node ID from the request packet is used as the destination node ID for the response packet. The response packet must be returned to the original source port. The path taken by the response is not necessarily the reverse of the path taken by the request. The network may be configured so that responses take very different paths than requests. If strict ordering is not required, the response, like the request, may use link-set adaptivity. The response packet is routed back to the SNID specified by the SID field of the request. The ACB field of the request packet is also duplicated for the response packet.

The response can be matched with the request using the TSN and the packet validity checks. If an ACK response passes these tests, the transaction layer passes the response data to the session layer, frees resources associated with the request, and reports the transaction as complete. If a NACK response passes these tests, the end node reports the failure of the transaction to the session layer. If a valid ACK/NACK response is not received within the allotted time limit, a time-out error is reported.

The requestor can stream many strictly ordered ServerNet II messages onto the wire before receiving an acknowledgment. The sliding window protocol allows the requestor to have up to 128 packets outstanding per VI.

The hardware can operate in one of two modes with respect to generating multiple outstanding request packets:

1. The hardware can stream packets from the same VI send queue onto the wire, and start the next descriptor before receiving all the acknowledgments from the current descriptor. This is referred to as "Next Descriptor After Launch" or NDAL.

2. The hardware can stream packets to a single descriptor onto the wire but wait for all the outstanding acknowledgments to complete before starting the next descriptor. This is referred to as "Next Descriptor after ACK" or NDAA.

The choice of NDAL or NDAA modes of operation is determined by how strongly ordered the packets are generated.

Ordered and unordered messages may be mixed on a single VI. When generating an unordered message, the requester must wait for completion of all acknowledgments to unordered packets before starting the next descriptor.

Ordering of Send Packets Presented to Transaction Layer

The VI architecture has no explicit ordering rules as to how the packets that make up a single descriptor are ordered among themselves. That is, VIA only guarantees the message ordering the client will see. For example, VIA requires that Send descriptors for a particular VI be completed in order, but the VIA specification does not say how the packets will proceed on the wire.

The ServerNet II SAN requires that all Send packets destined for a particular VI be delivered by the SAN in strict order. As long as deterministic routing is used, the network assures strict ordering along a path from a particular source node to a particular destination node. This is necessary because the receiving node places the incoming packets into a scatter list. Each incoming packet goes to a destination determined by the sum total of bytes of the previous packets. The strict ordering of packets is necessary to preserve integrity of the entire block of data being transferred because incoming packets are placed in consecutive locations within the block of data. Each packet has a sequence number to allow the receiver to detect an out of order, missing, or repeated packet.

There are two ways for an end node to meet these ordering requirements:

a. The end node can wait for the acknowledgment from each Send packet to complete before starting another Send packet for that VI. By waiting for each acknowledgment, the end node does not have to worry about the network providing strict ordering and can choose an arbitrary source port, adaptive link set and destination port for each message.

b. The end node can restrict all the Send operations for a given VI to use the same source port, the same destination port, and a single adaptive path. By choosing only one path through the network, the end node is guaranteed that each Send packet it launches into the network will arrive at the destination in order.

The second approach requires the VIA end node to maintain state per VI that indicates which source port destination port and adaptive path is currently in use for that particular VI. Furthermore, the second approach allows the hardware to process descriptors in the higher performance NDAL mode.

With the second approach, Send packets from a single VI can stream onto the network without waiting for their accompanying acknowledgments. An incrementing sequence number is used so the destination node can detect missing, repeated, or unordered Send packets.

Ordering of RDMA Packets

RDMA operations have slightly different ordering requirements than Send operations. An RDMA packet contains the address to which the destination end node writes the packet contents. This allows multiple RDMA packets within an RDMA message to complete out of order. The contents of each packet are written to the correct place in the end node's memory, regardless of the order in which they complete.

RDMA request packets may be sent ordered or unordered. A bit in the packet header is set to 1 for ordered packets and is set to 0 for unordered packets. As will be explained later, this bit is used by the responder logic to determine if it should increment its copy of the expected sequence number. Sequence numbers do not increment for unordered packets. The end node is free to use different source ports, destination ports and adaptive paths for the packets. This freedom can be exploited for a performance gain through multipathing; simultaneously sending the RDMA packets of a single message across multiple paths.

When RDMA Read or Write packets are sent over a path that does not exhibit strict ordering with the Send packets from the same VI, care must be taken when launching packets for the following message. The next message cannot be started until the last acknowledgment of the RDMA Read or Write operation successfully completes.

In other words, when multipathing is used to generate RDMA Read or Write requests, the hardware must operate in the NDAA mode. This ensures the RDMA Read or Write is completed before moving on to subsequent descriptors.

An end node may choose to send RDMA packets strictly ordered. This can be advantageous for smaller RDMA transfers as the hardware can operate in NDAL mode. The VI can proceed to the next descriptor immediately after launching the last packet of a message that is sent strictly ordered (and hence used incrementing sequence numbers).

Ordering of Generated Response Packets at the Responder

The ServerNet II end node must respond to incoming Send requests and RDMA Write requests from a particular VI in strict order, and must write these packets to memory in strict order.

The ServerNet II end node must also respond to incoming RDMA Read requests from a particular VI in strict order.

Because response packets are transported by the network in strict order, the requestor will receive all incoming response packets for a particular VI in the same order as that in which the corresponding requests were generated.

VIA Message Sequence Numbers

The ServerNet SAN uses acknowledgment packets to inform the requestor that a packet completed successfully. Sequence numbers in the packets (and acknowledgments) are used to. allow the sender to support multiple outstanding requests to ensure adequate performance and to be able to recover from errors occurring in the network.

FIG. 6 is a graph depicting the generation, checking, and updating of VIA sequence numbers at requestor and responder nodes. In FIG. 6, time increases in the downward direction. Requests are indicated by solid arrows directed to the right and responses by dotted arrows directed to the left.

Sequence Number Initialization

The requestor and responder logic each maintain an 8 bit sequence numbered for each VI in use. When the VI is created, the requester on one node and the respond on the remote node initialize their sequence numbers to a common value, zero i the preferred embodiment.

After this, the requester places its sequence number into each of the outgoing request packets. As depicted in FIG. 6, the sequence number, SEQ, is included in each request packet. The responder compares the sequence number from the incoming request packet with the responder's local copy.

The responder uses this comparison to determine if the packet is valid, if it is a duplicate of a packet already received, or if it is an out-of-sequence packet. An out-of-sequence packet can only happen if the responder missed an incoming packet. The responder can choose to return a 'sequence error NACK packet' or it can simply ignore the out-of-sequence packet. In the latter case, the requester will have a time-out on the request (and presumably on the packet the responder missed) and initiate error recover. Generating a sequence error NACK packet is preferred as it forces the requester to start error recovery more quickly.

The following describes how the sequence numbers are generated and checked.

Generating Sequence Numbers for Request Packets

When transmitting ordered packets (i.e. transfers are on a specific source port to a specific destination port and the ACB specifies a specific lane) the request sequence number is incremented after each packet is sent. When transmitting unordered packets (i.e. multipathing is used and/or the ACB bits specify full link set adaptivity) the request sequence number is not incremented after such a packet is sent.

For example, in FIG. 6, during the first two Send transactions, the local copy of the request sequence number is incremented after the packet is sent (Rqst. SN=0 to 6). For the RDMA operation, which sends 2500 bytes unordered, the requester does not increment local copy of the request sequence number (Rqst. SN=6). The requester does not increment the local copy of the SN until after the first packet of the Send following the RDMA is transmitted.

Send packets are typically sent fully ordered lest the requestor have to wait for an acknowledgment for each packet before proceeding to the next. On the other hand, RDMA packets may be sent either ordered or unordered. To take advantage of multipathing, a requestor must use unordered RDMA packets.

The sender guarantees to never exceed the windowsize number of packets outstanding per VI. If S is the number of bits in the sequence number, then the windowsize is 2**(S-1).

A packet is outstanding until it and all its predecessors are acknowledged. The requestor does not mark a descriptor done until all packets requested by that descriptor are positively acknowledged.

Checking Sequence Numbers on Incoming Request Packets

The destination node responding to the incoming request packet checks each incoming request packet to verify its sequence number against the responder's local copy.

The responder logic compares its sequence number with the packet's sequence number to determine if the incoming packet is either:

- the expected packet it is looking for (i.e., the packet's sequence number is the same as the sequence number maintained by the responder logic), in which case the responder processes the packet and if all other checks are passed, the packet is Acknowledged and committed to memory. If the transaction is ordered, the responder increments its sequence number. If the transaction is unordered, the responder does not increment its sequence number;
- an out-of-sequence packet (which means an earlier incoming packet must have gotten lost), beyond the one it is looking for in which case the responder NACKs the packet and throws it away. The receive logic in the VI is not stopped and the responder does not increment its sequence number; or
- a duplicate packet (which is being resent because the requestor must not have received an earlier ACK) in which case the responder ACKs the packet and throws it away. If the request had been an RDMA Read, the responder completes the read operation and returns the data with a positive acknowledgment.

An example of the responder checking sequence numbers for ordered and unordered packets is given in FIG. 6. In FIG. 6, during the first two Send transactions, the responder checks that the SEQ in the packet matches the local copy of Rsp. SN. Since the Send packets include ACB indicating ordered packets, the Rsp. SN is incremented after each response packet is transmitted. At the end of the first two Send transactions, Rqst. SN and Rsp. SN both equal 6. The packets for the RDMA include an ACB indicating unordered receipt is allowed. Neither the requestor or responder increments its local copy of SN. Thus, at the end of the RDMA transaction both Rqst. SN and Rsp. SN=6. The first packet of the subsequent Send transaction has SEQ=6 and SEQ matches the local copy of Rsp. SN. Since Send packets are ordered, the responder increments its local copy of Rsp. SN.

Sequence Numbers on Response Packets

When generating either a positive or negative acknowledgment, the responder logic copies the incoming sequence number and uses it in the sequence number field of the acknowledgment.

The requestor logic matches incoming responses with the originating request by comparing the SourceID, VI number, Sequence number, transaction type, and transaction Serial Number (TSN) with that of the originating request.

Error Recovery and Path State

Error recovery is initiated by the requesting node whenever the requestor fails to get a positive acknowledgment for each of its request packets. A time-out or NACK indicating a sequence number error, can cause the requestor's Kernel Agent to start error recovery.

Error recovery involves three basic steps:

1) Completing a barrier operation(s) to flush out any errant request or response packets.

2) Disabling a bad path if the barrier operation failed.

3) Retransmitting from the earliest packet that had failed.

The first two steps will now be described with reference to FIG. 7, which is two interlocked state diagrams showing the state that software on the requestor and responder moves through for each path. In FIG. 7, dashed lines represent Kernel to Kernel Supervisory Protocol messages that modify the remote node's state.

The ServerNet architecture allows multiple paths between end nodes. The requestor repeats these two basic steps on each path until the packet is transmitted successfully.

The requestor and responder SW each maintain a view of the state of each path. The requestor uses its view of the path state to determine which path it uses for Send and RDMA operations. The responder uses its view of the path state to determine which input paths it allows incoming requests on. The responder logic maintains a four bit field (ReqInPath Vector) for each VI in use. Each of the four bits corresponds to one of the four possible paths between the requestor's two ports and the responder's two ports. The requestor only

accepts incoming requests from a particular source or destination port if the corresponding bit in the ReqInPathVector is set

The requestor and responder communicate using the kernel-to-kernel Supervisory protocol to communicate path state changes.

The requester's view of the path state transitions from good to bad whenever the requestor fails to get an acknowledgment (either positive or negative) to a request. The requestor detects the lack of an ACK or NACK by getting a time-out error. The requestor can attempt a barrier operation on the path to see if the failure is permanent or transient. If the barrier succeeds, the path is considered good and the original operation can be retried. If the barrier fails, the requestor must resort to a different good path.

Before the requestor can try a different good path, the requestor must inform the destination that the original path is bad. This is done by any path possible. For example, in VIA the Kernel Agent to Kernel Agent Supervisory Protocol is used. After the destination is informed the path is bad, the destination disables a bit in a four bit field (ReqInPath Vector), thereby ignoring incoming requests from that path. The requester then stops using the bad path until a subsequent barrier transaction determines that the path is good. After the destination acknowledges the supervisory protocol message, indicating that the destination has disabled requests from the offending path, the requestor is free to retry the message on a different path.

After a time-out error, the requester attempts to bring the path back to a useful state by completing a barrier operation. The barrier operation ensures there are no other packets in any buffer that might show up later and corrupt the data transfer.

Barrier operations are used in error recovery to flush any stale request or stale response packets from a particular path in the SAN. A path is the collection of ServerNet links between a specific port of two end nodes.

A VIA barrier operation is done with a RDMA Write followed by an RDMA Read. A number chosen from a large number space (either incrementing or pseudo random) is written to a fixed location (e.g. a page number agreed to, a priori, by the kernel agent-to-kernel agent Supervisory Protocol and either a fixed or random offset within the page). The number is then read back with an RDMA read. If the read value matches the write value, then the barrier succeeded and there are guaranteed to be no more Send or RDMA request or response packets on that path between the requester and responder.

If the RDMA operation fails because the number read back does not match the number written, then the barrier is tried again. This could have happened because a previous response in the network came back and fulfilled the barrier.

Note that the barrier needs to be done separately on all paths the RDMA operation could have taken. That is, if the RDMA operation was being generated from multiple source ports (multipathing) and was using full link adaptivity (the packets were allowed to take any one of four possible "lanes"), then separate barrier operations must be done from each source port to each destination port, over each of the possible link adaptive paths.

The barrier operation must be done for each of the possible "lanes" between a specific Source port and Destination port. A barrier done on one VI ensures that all other VIs using that source port and destination port have no remaining request or response packets lurking in the SAN.

If a path traverses a "fat-pipe" a separate barrier must be sent down each lane of the fat pipe. SW can either blindly send four barrier operations (one for each lane) or it can maintain state, telling how many lanes are in use on the fat pipes between any given source/destination pair. The barrier need only be sent along the same path as the original request that failed. There is no requirement for the barrier to be sent from or to the same VI.

Turning now to the third step, i.e., retransmitting from the earliest packet that failed to receive a positive acknowledgement.[, after] After notification of the error, requestor retransmits the packets starting at (or before) the packet that failed to receive a positive acknowledgement. The requestor can restart up to WindowSize number of packets. The responder logic acknowledges and then ignores any packets that are resent if they have already been stored in the receive queue. When the correct packet is reached, the responder logic can tell from the sequence number that it is now time to resume writing the data to the receive queue.

Examples of retransmission after failure to receive a response are depicted in FIGS. 8 and 9. In FIG. 8, the request packet with SEQ=2 is corrupted. The missing request is detected by the responder on the next packet and NACKed (Negative Acknowledged) and all subsequent packets are thrown away and NACKed. The requestor resets its send engine to start generating packets at the one that failed to receive an ACK (in this case Rqst. SN=2). The responder recognizes the SEQ=2 and accepts the packets.

In FIG. 9, the response packet with SEQ=1 is corrupted. The missing response is detected when the requestor times out its transaction. The requestor resets its send engine to start generating packets at the one that failed to receive an ACK (in this case rqst.SN=1). The responder recognizes the resent packets as already having been received, acknowledges them and discards the data.

Note that the response packets for this particular RDMA transaction all have the same value of SEQ because the request SNs and response SNs are not incremented for RDMA transactions that are unordered. In this case, the TSNs are utilized by the requestor to match response packets to outstanding requests.

Error recovery places several requirements on the requestor's KA (Kernel Agent, the kernel mode driver code responsible for SAN error recovery):

1) The KA must determine the sequence number to restart with.

2) The KA must determine the proper data contents of the packet to be resent.

3) In order for the KA to determine the appropriate sequence number, it must be aware of how the hardware packetizes data under any given combination of descriptors, data segments, page crossings etc.

Note the responder side does not require KA involvement (unless a barrier operation fails).

The invention has now been described with reference to the preferred embodiments. Alternatives and substitutions will now be apparent to persons of skill in the art. For example, while the invention has been described in the context of the ServerNet II SAN, the principles of the invention are useful in any network that utilizes multiple paths between end nodes. Accordingly, it is not intended to limit the invention except as provided by the appended claims.

What is claimed is:

1. A method for error detection and recovery in a system with a plurality of networked nodes, including source and destination, communicating with each other via paths, comprising:

creating a request packet for a request transaction,

routing the request packet from the source to the destination via a particular path;

maintaining at each of the source and destination a status for each of its respective paths;

detecting a time-out error for failure within a predetermined time limit to receive at the source an acknowledge (ACK) packet or a negative-acknowledge (NACK) packet in response to the request packet, the time-out error created from a failure of the particular path;

performing a barrier transaction via the particular path to determine if the failure of the particular path is transient or permanent;

periodically repeating the barrier transaction via the particular path in order to determine if its failure is cured;

re-transmitting the request packet via the particular path if the failure is transient; and

if the failure is permanent,

updating at the source the status for the particular path, and

routing to the destination information about the updated status via an alternate path to prompt updating at the destination of the status for the particular pat wherein a failed path is not used.

2. The method of claim **1**, wherein when the transaction is ordered the routing is deterministic preserving strict ordering of request, ACK and NACK packets by selecting a single path as the particular path.

3. A method as in claim **1**, wherein when the transaction is not ordered the routing is link set adaptive enabling dynamic change of request, ACK and NACK packet routing by selection of paths from a link set.

4. A method as in claim **1**, wherein the request packet is discarded if it is invalid.

5. A method as in claim **1**, wherein the request packet contains a sequence number to indicate its place in a sequence of request packets and adaptive control bits to indicate the routing as ordered or not ordered.

6. A method as in claim **1**, further comprising:

maintaining a sequence number in each of the source and destination, the sequence numbers being initialized to a common value; and

including the sequence number in the request packet, wherein the sequence number is tracked by the source and destination, and wherein the NACK packet is created if a mismatch is found between the sequence number in the request packet and the sequence number at the destination.

7. A method for error detection and recovery in a system with a plurality of networked nodes, including source and destination, communicating with each other via paths, comprising:

maintaining a sequence number in each of the source and destination, the sequence numbers being initialized to a common value;

creating a request packet for a request transaction, the request packet containing the sequence number;

routing the request packet from the source to the destination via a particular path, wherein if the request transaction is ordered, the sequence number at the source is incremented;

maintaining at each of the source and destination a status for each of its respective paths;

checking the request packet for integrity and, if the request packet is valid, matching the sequence number in the packet with the sequence number at the destination;

creating an acknowledge (ACK) packet for a response transaction if the request packet is valid and the sequence number matching succeeds, and routing the ACK packet to the source, the ACK packet containing the sequence number from the request packet;

creating a negative acknowledge (NACK) packet for the response transaction if the sequence number matching fails, and routing the NACK packet to the source, the NACK packet containing the sequence number from the request packet;

incrementing the sequence number at the destination if the request transaction is ordered and the sequence number matching succeeds;

detecting a time-out error for failure within a predetermined time limit to receive at the source the ACK or the NACK packets in response to the request packet, the time-out error created from a failure of the particular path;

performing a barrier transaction via the particular path to determine if the failure of the particular path is transient or permanent;

periodically repeating the barrier transaction via the particular path in order to determine if its failure is cured;

re-transmitting the request packet via the particular path if the failure is transient; and

if the failure is permanent, updating a status at the source for the particular path and routing to the destination information about the updated status via an alternate path to prompt updating of the status for the particular path at the destination, wherein a failed path is not used.

8. A method as in claim **7**, wherein when the transaction is ordered the routing is deterministic preserving strict ordering of packets by selecting a single path as the particular path.

9. A method as in claim **7**, wherein when the transaction is not ordered the routing is link set adaptive enabling dynamic change of packet routing by selection of paths from a path set.

10. A method as in claim **7**, wherein the request packet is discarded if it is invalid.

11. A method as in claim **7**, wherein the request packet contains adaptive control bits indicating whether the transaction is ordered or not ordered.

12. A system for error detection and recovery with a plurality of networked nodes, including source and destination, communicating with each other via paths, comprising:

means for creating a request packet for a request transaction; path means for routing the request packet from the source to the destination via a particular path;

means for maintaining at each of the source and destination a status for each of its respective paths;

means for detecting a time-out error for failure within a predetermined time limit to receive at the source an acknowledge (ACK) packet or a negative-acknowledge (NACK) packet in response to the request packet, the time-out error created from a failure of the particular path;

means for performing a barrier transaction via the particular path to determine if the failure of the particular path is transient or permanent, including means for periodically repeating the barrier transaction via the particular path in order to determine if its failure is cured;

means for re-transmitting from the source the request packet via the particular path if the failure is transient; and

if the failure is permanent,

means for updating at the source the status for the particular path, and

means for routing to the destination information about the updated status via an alternate path to prompt updating at the destination of the status for the particular path, wherein a failed path is not used.

13. The system of claim 12, wherein when the transaction is ordered the routing is deterministic preserving strict ordering of request, ACK and NACK packets by selecting a single path as the particular path.

14. A system as in claim 12, wherein when the transaction is not ordered the routing is link set adaptive enabling dynamic change of request, ACK and NACK packet routing by selection of paths from a link set.

15. A system as in claim 12, further comprising:

means for discarding the request packet if it is invalid.

16. A system as in claim 12, wherein the request packet contains a sequence number to indicate its place in a sequence of request packets and adaptive control bits to indicate the routing as ordered or not ordered.

17. A system as in claim 12, further comprising:

means for maintaining a sequence number in each of the source and destination, the sequence numbers being initialized to a common value; and

means for including the sequence number in the request packet, wherein the sequence number is tracked by the source and destination, and wherein the NACK packet is created if a mismatch is found between the sequence number in the request packet and the sequence number at the destination.

18. A system for error detection and recovery in a system with a plurality of networked nodes, including source and destination, communicating with each other via paths, comprising:

means for maintaining a sequence number in each of the source and destination, the sequence numbers being initialized to a common value;

means for creating a request packet for a request transaction, the request packet containing the sequence number;

means for routing the request packet from the source to the destination via a particular path, including means for incrementing the sequence number at the source if the request transaction is ordered;

means for maintaining at each of the source and destination a status for each of its respective paths;

means for checking the request packet for integrity, including means for matching the sequence number in the packet against the sequence number at the destination if the request packet is valid;

means for creating an acknowledge (ACK) packet for a response transaction if the request packet is valid and a sequence number matching succeeds, including means for routing the ACK packet to the source, the ACK packet containing the sequence number from the request packet;

means for creating a negative acknowledge (NACK) packet for the response transaction if the sequence number matching fails, including means for routing the NACK packet to the source, the NACK packet containing the sequence number from the request packet;

means for incrementing the sequence number if the request transaction is ordered and the sequence number matching succeeds;

means for detecting a time-out error for failure within a predetermined time limit to receive at the source the ACK or the NACK packets in response to the request packet, the time-out error created from a failure of the particular path;

means for performing a barrier transaction via the particular path to determine if the failure of the particular path is transient or permanent, including by periodically repeating the barrier transaction via the particular path in order to determine if its failure is cured;

means for re-transmitting the request packet via the particular path if the failure is transient; and

if the failure is permanent,

means for updating a status at the source for the particular path, and

means for routing to the destination information about the updated status via an alternate path to prompt updating of the status for the particular path at the destination, wherein a failed path is not used.

19. A system as in claim 18, wherein when the transaction is ordered the routing is deterministic preserving strict ordering of request, ACK and NACK packets by selecting a single path as the particular path.

20. A system as in claim 18, wherein when the transaction is not ordered the routing is link set adaptive enabling dynamic change of request, ACK and NACK packet routing by selection of paths from a path set.

21. A system as in claim 18, further comprising:

means for discarding the request packet if it is invalid.

22. A system as in claim 18, wherein the request packet contains adaptive control bits indicating whether the transaction is ordered or not ordered.

* * * * *

US006119244A

United States Patent [19]

Schoenthal et al.

[11] Patent Number: 6,119,244

[45] Date of Patent: Sep. 12, 2000

[54] COORDINATING PERSISTENT STATUS INFORMATION WITH MULTIPLE FILE SERVERS

[75] Inventors: Scott Schoenthal, San Ramon; Alan Rowe, San Jose; Steven R. Kleiman, Los Altos, all of Calif.

[73] Assignee: Network Appliance, Inc.

[21] Appl. No.: 09/139,257

[22] Filed: Aug. 25, 1998

[51] Int. Cl.[7] G06F 11/14

[52] U.S. Cl. 714/4; 709/216; 714/12

[58] Field of Search 714/4, 5, 6, 7, 714/8, 15, 18, 10, 11, 12, 13; 709/213, 214, 215, 216

[56] References Cited

U.S. PATENT DOCUMENTS

4,351,023	9/1982	Richer	714/13
4,825,354	4/1989	Agrawal et al.	364/200
4,984,272	1/1991	McIlroy et al.	380/25
5,113,442	5/1992	Moir	380/25
5,144,659	9/1992	Jones	380/4
5,146,588	9/1992	Crator	714/6
5,202,983	4/1993	Orita et al.	395/600
5,222,217	6/1993	Blount et al.	395/325
5,261,051	11/1993	Masden et al. .	
5,283,830	2/1994	Hinsley et al.	380/25

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

0306244A2	3/1989	European Pat. Off. .
0308056A2	3/1989	European Pat. Off. .
0410630A	1/1991	European Pat. Off. .
0 477 039	3/1992	European Pat. Off. .
0566967A	10/1993	European Pat. Off. .

(List continued on next page.)

OTHER PUBLICATIONS

Kleiman "Using NUMA Interconnects for Highly Available Filers" IEEE Micro 1999, pp. 42–48.

"Mapping the VM text files to the AIX text files", IBM Technical Disclosure Bulletin., vol. 33, No. 2, Jul. 1990 (1900–07), p. 341 XP000123641, IBM Corp. New York., US ISSN: 0018–8689—the whole document.

"Migrated Data Backup Utility", IBM Technical Disclosure Bulletin., vol. 37, No. 06B, Jun. 1994 (1994–06), pp. 505–507, XP000456079, IBM Corp. New York., US ISSN: 0018–8689.

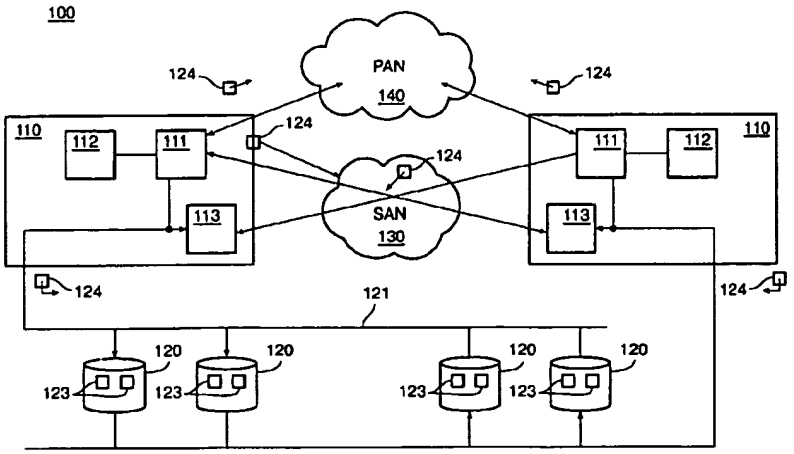
(List continued on next page.)

Primary Examiner—Robert W. Beausoliel, Jr.
Assistant Examiner—Bryce Bonzo
Attorney, Agent, or Firm—Swernofsky Law Group

[57] ABSTRACT

The invention provides a storage system, and a method for operating a storage system, that provides for relatively rapid and reliable takeover among a plurality of independent file servers. Each file server maintains a reliable communication path to the others. Each file server maintains its own state in reliable memory. Each file server regularly confirms the state of the other file servers. Each file server labels messages on the redundant communication paths, so as to allow other file servers to combine the redundant communication paths into a single ordered stream of messages. Each file server maintains its own state in its persistent memory and compares that state with the ordered stream of messages, so as to determine whether other file servers have progressed beyond the file server's own last known state. Each file server uses the shared resources (such as magnetic disks) themselves as part of the redundant communication paths, so as to prevent mutual attempts at takeover of resources when each file server believes the other to have failed. Each file server provides a status report to the others when recovering from an error, so as to prevent the possibility of multiple file servers each repeatedly failing and attempting to seize the resources of the others.

26 Claims, 2 Drawing Sheets



U.S. PATENT DOCUMENTS

5,335,235	8/1994	Arnott	714/801
5,357,612	10/1994	Alaiwan	709/216
5,454,095	9/1995	Kraemer	701/104
5,504,883	4/1996	Coverston et al. .	
5,572,711	11/1996	Hirsch et al.	395/500
5,617,568	4/1997	Ault et al.	395/612
5,668,958	9/1997	Bendert et al.	710/128
5,675,782	10/1997	Montague et al.	395/609
5,689,701	11/1997	Ault et al.	395/610
5,720,029	2/1998	Kern .	
5,737,523	4/1998	Callaghan et al.	395/187.01
5,761,669	6/1998	Montague et al.	707/103
5,819,292	10/1998	Hitz et al.	707/203
5,825,877	10/1998	Dan et al.	380/4
5,876,278	3/1999	Cheng	454/184
5,890,959	4/1999	Pettit et al.	454/184
5,915,087	6/1999	Hammond et al.	395/187.01
5,931,935	8/1999	Cabrera et al.	710/260
5,963,962	10/1999	Hitz et al.	707/202

FOREIGN PATENT DOCUMENTS

629956 A2	12/1994	European Pat. Off. .	
629956 A3	12/1994	European Pat. Off. .	
747829	12/1996	European Pat. Off. .	
756235	1/1997	European Pat. Off. .	
0760503A1	3/1997	European Pat. Off. .	
1-273395	11/1989	Japan	361/695
WO 9821656	5/1998	WIPO .	

OTHER PUBLICATIONS

Borr A J: "SecureShare: safe Unix/Windows file sharing through multiprotocol locking" Proceeding of the 2nd Usenix Windows NT Symposiom, proceedings of 2nd Usenix Windows NT Symposiom, Seattle, WA, USA, Aug. 3–5, 1998, pp. 117–126, XP002097387 ISBN 1–880446–95–2, 1998, Berkeley, CA, USA, Usenix Assoc. USA.

Tanner J:"CIFS: Common Internet File System" Unix Review, vol. 31, Feb. 1997, pp. 31/32, 34, XPOOO783952 see whole document, relevant to claim No. 1–38.

R. Reichel: "Inside Windows NT Security: Part 1" Windows/DOS Developers' Journal, vol. 4, No. 4, Apr. 1993, pp. 6–19, XP002107445, Lawrence, KS, USA.

R. Nass: Connect Disk Arrays to Eisa or PCI Buses Electronic Design, vol. 41, No. 23, Nov. 1993, Cleveland, OH, USA pp. 152–154, XP000417908 see p. 152–P. 153, right–hand column, line 10.

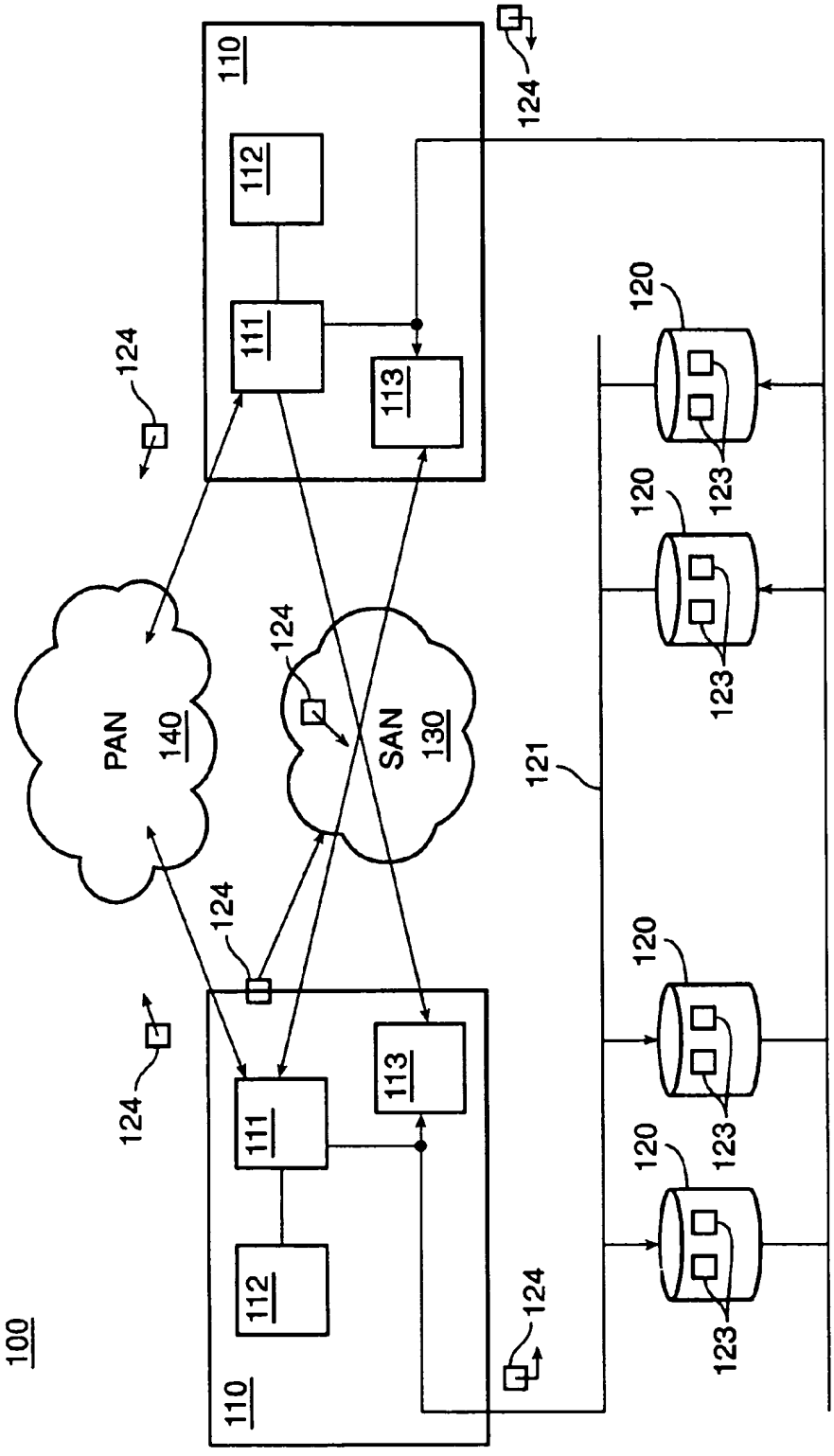


FIG. 1

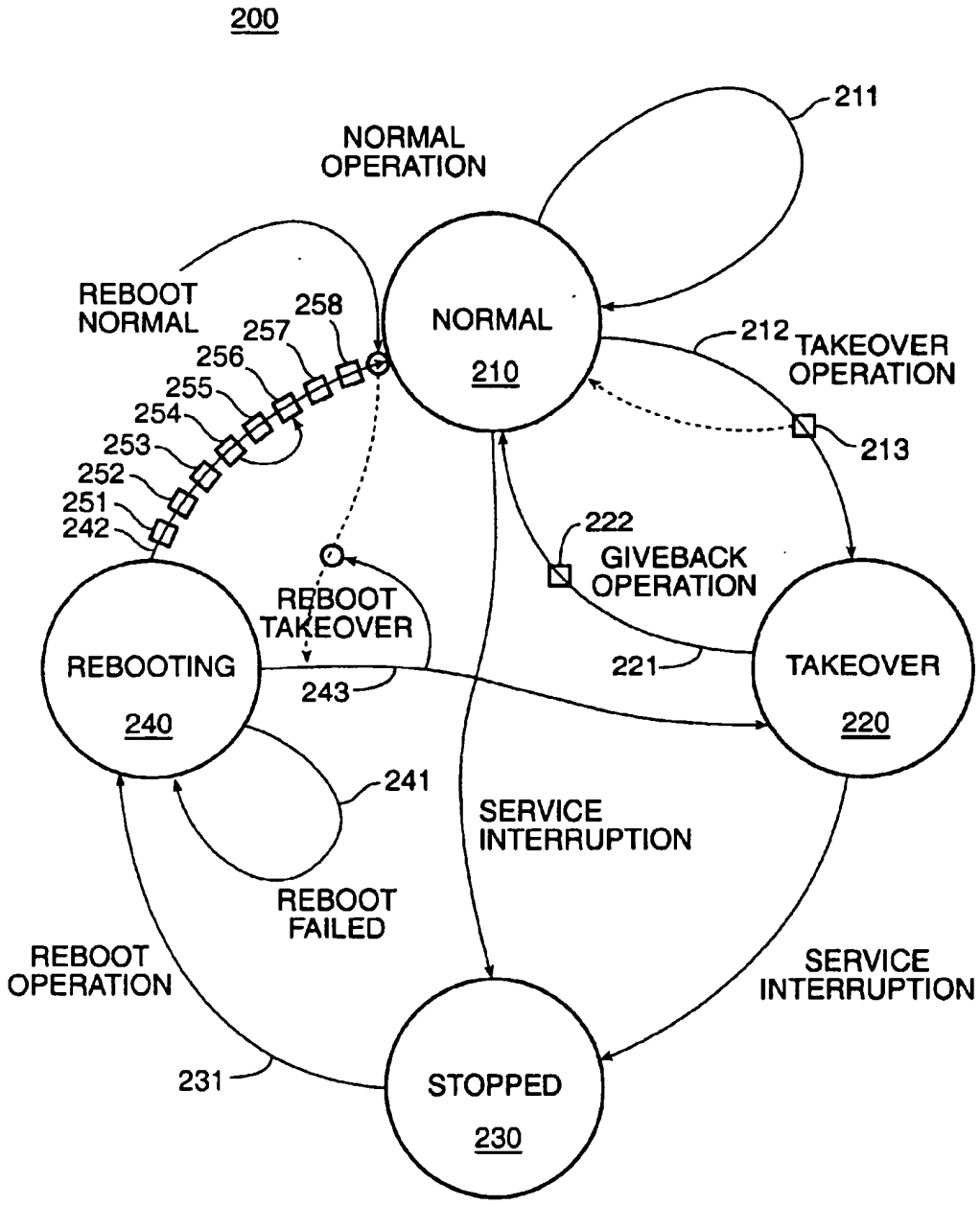

200
211
NORMAL OPERATION
REBOOT NORMAL
258
257
256
255
254
253
252
251
242
NORMAL
210
212
TAKEOVER OPERATION
213
222
GIVEBACK OPERATION
REBOOT TAKEOVER
REBOOTING
240
243
221
TAKEOVER
220
241
SERVICE INTERRUPTION
REBOOT FAILED
REBOOT OPERATION
SERVICE INTERRUPTION
231
STOPPED
230

FIG. 2

COORDINATING PERSISTENT STATUS INFORMATION WITH MULTIPLE FILE SERVERS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to computer systems.

2. Related Art

Computer storage systems are used to record and retrieve data. It is desireable for the services and data provided by the storage system to be available for service to the greatest degree possible. Accordingly, some computer storage systems provide a plurality of file servers, with the property that when a first file server fails, a second file server is available to provide the services; and the data otherwise provided by the first. The second file server provides these services and data by takeover of resources otherwise managed by the first file server.

One problem in the known art is that when two file servers each provide backup for the other, it is important that each of the two file servers is able to reliably detect failure of the other and to smoothly handle any required takeover operations. It would be advantageous for this to occur without either of the two file servers interfering with proper operation of the other. This problem is particularly acute in systems when one or both file servers recover from a service interruption.

Accordingly, it would be advantageous to provide a storage system and a method for operating a storage system, that provides for relatively rapid and reliable takeover among a plurality of independent file servers. This advantage is achieved in an embodiment of the invention in which each file server (a) maintains redundant communication paths to the others, (b) maintains its own state in persistent memory at least some of which is accessible to the others, and (c) regularly confirms the state of the other file servers.

SUMMARY OF THE INVENTION

The invention provides a storage system and a method for operating a storage system, that provides for relatively rapid and reliable takeover among a plurality of independent file servers. Each file server maintains a reliable (such as redundant) communication path to the others, preventing any single point of failure in communication among file servers. Each file server maintains its own state in reliable (such as persistent) memory at least some of which is accessible to the others, providing a method for confirming that its own state information is up to date, and for reconstructing proper state information if not. Each file server regularly confirms the state of the other file servers, and attempts takeover operations only when the other file servers are clearly unable to provide their share of services.

In a preferred embodiment, each file server sequences messages on the redundant communication paths, so as to allow other file servers to combine the redundant communication paths into a single ordered stream of messages. Each file server maintains its own state in its persistent memory and compares that state with the ordered stream of messages, so as to determine whether other file servers have progressed beyond the file server's own last known state. Each file server uses the shared resources (such as magnetic disks) themselves as part of the redundant communication paths, so as to prevent mutual attempts at takeover of resources when each file server believes the other to have failed.

In a preferred embodiment, each file server provides a status report to the others when recovering from an error, so as to prevent the possibility of multiple file servers each repeatedly failing and attempting to seize the resources of the others.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows a block diagram of a multiple file server system with coordinated persistent status information.

FIG. **2** shows a state diagram of a method of operation for a multiple file server system with coordinated persistent status information.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the following description, a preferred embodiment of the invention is described with regard to preferred process steps and data structures. However, those skilled in the art would recognize, after perusal of this application, that embodiments of the invention may be implemented using one or more general purpose processors (or special purpose processors adapted to the particular process steps and data structures) operating under program control, and that implementation of the preferred process steps and data structures described herein using such equipment would not require undue experimentation or further invention.

In a preferred embodiment, the file server system, and each file server therein, operates using inventions described in the following patent applications:

- application Ser. No. 09/037,652, filed Mar. 10, 1998, in the name of inventor Steven Kleiman, titled "Highly Available File Servers," attorney docket number NAP-012.

Each of these applications is hereby incorporated by reference as if fully set forth herein. They are collectively referred to as the "Clustering Disclosures."

In a preferred embodiment, each file server in the file server system controls its associated mass storage devices so as to form a redundant array, such as a RAID storage system, using inventions described in the following patent applications:

- application Ser. No. 08/471,218, filed Jun. 5, 1995, in the name of inventors David Hitz et al., titled "A Method for Providing Parity in a Raid Sub-System Using Non-Volatile Memory", attorney docket number NET-004;
- application Ser. No. 08/454,921, filed May 31, 1995, in the name of inventors David Hitz et al., titled "Write Anywhere File-System Layout", attorney docket number NET-005;
- application Ser. No. 08/464,591, filed May 31, 1995, in the name of inventors David Hitz et al., titled "Method for Allocating Files in a File System Integrated with a Raid Disk Sub-System", attorney docket number NET-006.

Each of these applications is hereby incorporated by reference as if fully set forth herein. They are collectively referred to as the "WAFL Disclosures."

System Elements

FIG. **1** shows a block diagram of a multiple file server system with coordinated persistent status information.

A system **100** includes a plurality of file servers **110**, a plurality of mass storage devices **120**, a SAN (system area network) **130**, and a PN (public network) **140**.

In a preferred embodiment, there are exactly two file servers **110**. Each file server **110** is capable of acting

independently with regard to the mass storage devices **120**. Each file server **110** is disposed for receiving file server requests from client devices (not shown), for performing operations on the mass storage devices **120** in response thereto, and for transmitting responses to the file server requests to the client devices.

For example, in a preferred embodiment, the file servers **110** are each similar to file servers described in the Clustering Disclosures.

Each of the file servers **110** includes a processor **111**, program and data memory **112**, and a persistent memory **113** for maintaining state information across possible service interruptions. In a preferred embodiment, the persistent memory **113** includes a nonvolatile RAM.

The mass storage devices **120** preferably include a plurality of writeable magnetic disks, magneto-optical disks, or optical disks. In a preferred embodiment, the mass storage devices **120** are disposed in a RAID configuration or other system for maintaining information persistent across possible service interruptions.

Each of the mass storage devices **120** are coupled to each of the file servers **110** using a mass storage bus **121**. In a preferred embodiment, each file server **110** has its own mass storage bus **121**. The first file server **110** is coupled to the mass storage devices **120** so as to be a primary controller for a first subset of the mass storage devices **120** and a secondary controller for a second subset thereof. The second file server **110** is coupled to the mass storage devices **120** so as to be a primary controller for the second subset of the mass storage devices **120** and a secondary controller for the first subset thereof.

The mass storage bus **121** associated with each file server **110** is coupled to the processor **111** for that file server **110** so that file server **110** can control mass storage devices **120**. In alternative embodiments, the file servers **110** may be coupled to the mass storage devices **120** using other techniques, such as fiber channel switches or switched fabrics.

The mass storage devices **120** are disposed to include a plurality of mailbox disks **122**, each of which has at least one designated region **123** into which one file server **110** can write messages **124** for reading by the other file server **110**. In a preferred embodiment, there is at least one designated region **123**, on each mailbox disk **122** for reading and at least one designated region **123** for writing, by each file server **110**.

The SAN **130** is coupled to the processor **111** and to the persistent memory **113** at each of the file servers **110**. The SAN **130** is disposed to transmit messages **124** from the processor **111** at the first file server **110** to the persistent memory **113** at the second file server **110**. Similarly, the SAN **130** is disposed to transmit messages **124** from the processor **111** at the second file server **110** to the persistent memory **113** at the first file server **110**.

In a preferred embodiment, the SAN **130** comprises a ServerNet connection between the two file servers **110**. In alternative embodiments, the persistent memory **112** may be disposed logically remote to the file servers **110** and accessible using the SAN **130**.

The PN **140** is coupled to the processor **111** at each of the file servers **110**. The PN **140** is disposed to transmit messages **124** from each file server **110** to the other file server **110**.

In a preferred embodiment, the PN **140** can comprise a direct communication channel, a LAN (local area network), a WAN (wide area network), or some combination thereof.

Although the mass storage devices **120**, the SAN **130**, and the PN **140** are each disposed to transmit messages **124**, the messages **124** transmitted using each of these pathways between the file servers **110** can have substantially differing formats, even though payload for those messages **124** is identical.

METHOD OF OPERATION

FIG. **2** shows a state diagram of a method of operation for a multiple file server system with coordinated persistent status information.

A state diagram **200** includes a plurality of states and a plurality of transitions therebetween. Each transition is from a first state to a second state and occurs upon detection of a selected event.

The state diagram **200** is followed by each of the file servers **110** independently. Thus, there is a state for "this" file server **110** and another (possibly same, possibly different) state for the "the other" file server **110**. Each file server **110** independently determines what transition to follow from each state to its own next state. The state diagram **200** is described herein with regard to "this" file server **110**.

In a NORMAL state **210**, this file server **110** has control of its own assigned mass storage devices **120**.

In a TAKEOVER state **220**, this file server **110** has taken over control of the mass storage devices **120** normally assigned to the other file server **110**.

In a STOPPED state **230**, this file server **110** has control of none of the mass storage devices **120** and is not operational.

In a REBOOTING state **240**, this file server **110** has control of none of the mass storage devices **120** and is recovering from a service interruption.

NORMAL State

In the NORMAL state **210**, both file servers **110** are operating properly, and each controls its set of mass storage devices **120**.

In this state, each file server **110** periodically sends state information in messages **124** using the redundant communication paths between the two file servers **110**. Thus, each file server **110** periodically transmits messages **124** having state information by the following techniques:

Each file server **110** transmits a message **124** by copying that message to the mailbox disks on its assigned mass storage devices **120**.

In a preferred embodiment, messages **124** are transmitted using the mailbox disks by writing the messages **124** to a first mailbox disk and then to a second mailbox disk.

Each file server **110** transmits a message **124** by copying that message **124**, using the SAN **130**, to its persistent memory **113** (possibly both its own persistent memory **113** and that for the other file server **110**).

In a preferred embodiment, messages **124** are transmitted using the SAN **130** using a NUMA technique.

and

Each file server **110** transmits a message **124** by transmitting that message **124**, using the PN **140**, to the other file server **110**.

In a preferred embodiment, messages **124** are transmitted using the PN **140** using encapsulation in a communication protocol known to both file servers **110**, such as UDP or IP.

Each message **124** includes the following information for "this" file server **110** (that is, the file server **110** transmitting the message **124**):

a system ID for this file server **110**;

a state indicator for this file server **110**;

In a preferred embodiment, the state indicator can be one of the following:

(NORMAL) operating normally,

(TAKEOVER) this file server **110** has taken over control of the mass storage devices **120**,

(NO-TAKEOVER) this file server **110** does not want the receiving file server to take over control of its mass storage devices **120**, and

(DISABLE) takeover is disabled for both file servers **110**.

a generation number Gi, comprising a monotonically increasing number identified with a current instantiation of this file server **110**;

In a preferred embodiment, the instantiation of this file server **110** is incremented when this file server **110** is initiated on boot-up. If any file server **110** suffers a service interruption that involves reinitialization, the generation number Gi will be incremented, and the message **124** will indicate that it is subsequent to any message **124** send before the service interruption.

and

a sequence number Si, comprising a monotonically increasing number identified with the current message **124** transmitted by this file server **110**.

Similarly, each message **124** includes the following information for "the other" file server **110** (that is, the file server **110** receiving the message **124**):

a generation number Gi, comprising a monotonically increasing number identified with a current instantiation of the other file server **110**;

and

a sequence number Si, comprising a monotonically increasing number identified with the most recent message **124** received from the other file server **110**.

Each message **124** also includes a version number of the status protocol with which the message **124** is transmitted.

Since the file server **110** receives the messages **124** using a plurality of pathways, it determines for each message **124** whether or not that message **124** is "new" (the file server **110** has not seen it before), or "old" (the file server **110** has seen it before). The file server **110** maintains a record of the generation number Gi and the sequence number Si of the most recent new message **124**. The file server **110** determines that the particular message **124** is new if and only if:

its generation number Gi is greater than the most recent new message **124**;

or

its generation number Gi is equal to the most recent new message **124** and its sequence number Si is greater than most recent new message **124**.

If either of the file servers **110** determines that the message **124** is not new, that file server **110** can ignore that message **124**.

In this state, each file server **110** periodically saves its own state information using the messages **124**. Thus, each file server **110** records its state information both on its own mailbox disks and in its own persistent memory **113**.

In this state, each file server **110** periodically watches for a state change in the other file server **110**. The first file server **110** detects a state change in the second file server **110** in one of at least two ways:

The first file server **110** notes that the second file server **110** has not updated its state information (using a message **124**) for a timeout period.

In a preferred embodiment, this timeout period is two-half seconds for communication using the mailbox disks and one-half second for communication using the SAN **130**. However, there is no particular requirement for using these timeout values; in alternative embodiments, different timeout values or techniques other than timeout periods may be used.

and

The first file server **110** notes that the second file server **110** has updated its state information (using one or more messages **124**) to indicate that the second file server **110** has changed its state.

In a preferred embodiment, the second file server **110** indicates when it is in one of the states described with regard to each message **124**.

If the first file server **110** determines that the second file server **110** is also in the NORMAL state, the NORMAL-OPERATION transition **211** is taken to remain in the state **210**.

The first file server **110** makes its determination responsive to messages **124** it receives from the second file server **110**. If there are no such messages **124** for a time period responsive to the timeout period described above (such as two to five times the timeout period), the first file server **110** decides that the second file server **110** has suffered a service interruption.

If the first file server **110** determines that the second file server **110** has suffered a service interruption (that is, the second file server **110** is in the STOPPED state **230**), the TAKEOVER-OPERATION transition **212** is taken to enter the TAKEOVER state **220**.

The TAKEOVER-OPERATION transition **212** can be disabled by a message **124** state indicator such as DISABLE or NO-TAKEOVER.

In a preferred embodiment, either file server **110** can disable the TAKEOVER-OPERATION transition **212** responsive to (a) an operator command, (b) a synchronization error between the persistent memories **113**, or (c) any compatibility mismatch between the file servers **110**.

To perform the TAKEOVER-OPERATION transition **212**, this file server **110** performs the following actions at a step **213**:

This file server **110** sends the message **124** state indicator TAKEOVER to the other file server **110**, using including the reliable communication path (including the mailbox disks **122**, the SAN **130**, and the PN **140**).

This file server **110** waits for the other file server **110** to have the opportunity to receive and act on the TAKEOVER-OPERATION transition **212** (that is, to suspend its own access to the mass storage devices **120**.

This file server **110** issues disk reservation commands to the mass storage devices **120** normally assigned to the other file server **110**.

This file server **110** takes any other appropriate action to assure that the other file server **110** is passive.

If the takeover operation is successful, the TAKEOVER-OPERATION transition **212** completes and this file server enters the TAKEOVER state **220**. Otherwise (such as if takeover is disabled), this file server **110** returns to the NORMAL state **210**.

TAKEOVER State

In the TAKEOVER state **220**, this file server **110** is operating properly, but the other file server **110** is not. This file server **110** has taken over control of both its and the other's mass storage devices **120**.

In this state, this file server **110** continues to write messages **124** to the persistent memory **113** and to the mailbox disks **122**, so as to preserve its own state in the event of a service interruption.

In this state, this file server **110** continues to control all the mass storage devices **120**, both its own and those normally assigned to the other file server **110**, until this file server **110** determines that it should give back control of some mass storage devices **120**.

In a preferred embodiment, the first file server **110** makes its determination responsive to operator control. An operator for this file server **110** determines that the other file server **110** has recovered from its service interruption. The GIVEBACK-OPERATION transition **221** is taken to enter the NORMAL state **210**.

In alternative embodiments, the first file server **110** may make its determination responsive to messages **124** it receives from the second file server **110**. If the second file server **110** sends messages **124** indicating that it has recovered from a service interruption (that is, it is in the REBOOTING state **240**), the first file server **110** may initiate the GIVEBACK-OPERATION transition **221**.

To perform the GIVEBACK-OPERATION transition **221**, this file server **110** performs the following actions at a step **222**:

- This file server **110** releases its disk reservation commands to the mass storage devices **120** normally assigned to the other file server **110**.
- This file server **110** sends the message **124** state indicator NORMAL to the other file server **110**, including using the mailbox disks **122**, the. SAN **130**, and the PN **140**.
- This file server **110** disables the TAKEOVER-OPERATION transition **212** by the other file server **110** until the other file server **110** enters the NORMAL state **210**. This file server **110** remains at the step **222** until the other file server **110** enters the NORMAL state **210**.

When the giveback operation is successful, the GIVEBACK-OPERATION transition **221** completes and this file server enters the NORMAL state **210**.

STOPPED State

In the STOPPED state **230**, this file server **110** has control of none of the mass storage devices **120** and is not operational.

In this state, this file server **110** performs no operations, until this file server **110** determines that it reboot.

In a preferred embodiment, the first file server **110** makes its determination responsive to operator control. An operator for this file server **110** determines that it has recovered from its service interruption. The REBOOT-OPERATION transition **231** is taken to enter the REBOOTING state **240**.

In alternative embodiments, the first file server **110** may make its determination responsive to a timer or other automatic attempt to reboot. When this file server **110** determines that it has recovered from its service interruption, it attempts to reboot, and the REBOOT-OPERATION transition **231** is taken to enter the REBOOTING state **240**.

REBOOTING State

In the REBOOTING state **240**, this file server **110** has control of none of the mass storage devices **120** and is recovering from a service interruption.

In this state, the file server **110** attempts to recover from a service interruption.

If this file server **110** is unable to recover from the service interruption, the REBOOT-FAILED transition **241** is taken and this file server **110** remains in the REBOOTING state **240**.

If this file server **110** is able to recover from the service interruption, but the other file server **110** is in the TAKEOVER state **220**, the REBOOT-FAILED transition **241** is taken and this file server **110** remains in the REBOOTING state **240**. In this case, the other file server **110** controls the mass storage devices **120** normally assigned to this file server **110**, and this file server **110** waits for the GIVEBACK-OPERATION transition **221** before re-attempting to recover from the service interruption.

If this file server **110** is able to recover from the service interruption, and determines it should enter the NORMAL state **210** (as described below), the REBOOT-NORMAL transition **242** is taken and this file server **110** enters the NORMAL state **210**.

If this file server **110** is able to recover from the service interruption, and determines it should enter the TAKEOVER state **210** (as described below), the REBOOT-TAKEOVER transition **243** is taken and this file server **110** enters the TAKEOVER state **210**.

In a preferred embodiment, this file server **110** performs the attempt to recover from the service interruption with the following steps.

At a step **251**, this file server **110** initiates its recovery operation.

At a step **252**, this file server **110** determines whether it is able to write to any of the mass storage devices **120** (that is, if the other file server **110** is in the TAKEOVER state **220**). If so, this file server **110** displays a prompt to an operator so indicating and requesting the operator to command the other file server **110** to perform the GIVEBACK-OPERATION transition **221**.

This file server **110** waits until the operator commands the other file server **110** to perform a giveback operation, waits until the GIVEBACK-OPERATION transition **221** is complete, and proceeds with the next step.

At a step **253**, this file server **110** determines the state of the other file server **110**. This file server **110** makes this determination in response to its own persistent memory **113** and the mailbox disks **122**. This file server **110** notes the state it was in before entering the REBOOTING state **240** (that is, either the NORMAL state **210** or the TAKEOVER state **220**).

If this file server **110** determines that the other file server **110** is in the NORMAL state **210** it proceed s with the step **254**. If this file server **110** determines that it had previously taken over all the mass storage devices **120** (that is, that the other file server **110** is in the STOPPED state **230** or the REBOOTING state **240**), it proceeds with the step **255**.

At a step **254**, this file server **110** attempts to seize its own mass storage devices **120** but not those normally assigned to the other file server **110**. This file server **110** proceeds with the step **256**.

At a step **255**, this file server **110** attempts to seize both its own mass storage devices **120** and those normally assigned to the other file server **110**. This file server **110** proceeds with the step **256**.

At a step **256**, this file server **110** determines whether its persistent memory **113** is current with regard to pending file server operations. If not, this file server **110** flushes its persistent memory **113** of pending file server operations.

At a step **257**, this file server **110** determines if it is able to communicate with the other file server and if there is anything (such as an operator command) preventing takeover operations. This file server **110** makes its determination in response to the persistent memory **113** and the mailbox disks **122**.

At a step **258**, if this file server **110** was in the NORMAL state **210** before entering the REBOOTING state **240** (that is, this file server **110** performed the step **254** and seized only its own mass storage devices **120**), it enters the NORMAL state **210**.

At a step **258**, if this file server **110** was in the TAKEOVER state **220** before entering the REBOOTING state **240**

(that is, this file server **110** performed the step **255** and seized all the mass storage devices **120**, it enters the TAKEOVER state **220**.

ALTERNATIVE EMBODIMENTS

Although preferred embodiments are disclosed herein, many variations are possible which remain within the concept, scope, and spirit of the invention, and these variations would become clear to those skilled in the art after perusal of this application.

What is claimed is:

1. A file server including

a set of storage devices capable of being shared with a second file server;

a controller disposed for coupling to said shared set of storage devices;

a transceiver disposed for coupling to a communication path and for communicating messages using said communication path, said communication path using said shared set of storage devices to communicate said messages;

a takeover monitor coupled to at least part of said shared set of storage devices, and responsive to said communication path and said shared set of storage devices.

2. A file server as in claim **1**, including persistent memory storing state information about said file server, said takeover monitor being responsive to said persistent memory.

3. Apparatus including

a shared resource;

a pair of servers each coupled to said shared resource and each disposed for managing at least part of said shared resource;

a communication path disposed for coupling a sequence of messages between said pair, said communication path disposed for using s aid shared resource for coupling said sequence of messages;

each one of said pair being disposed for takeover of at least part of said shared resource in response to said communication path;

whereby said communication path prevents both of said pair from concurrently performing said takeover.

4. Apparatus as in claim **3**, wherein

at least one said server includes a file server;

said shared resource includes a storage medium; and

said communication path includes a designated location on said storage medium.

5. Apparatus as in claim **3**, wherein

each one of said pair includes persistent memory;

said persistent memory being disposed for storing state information about said pair; and

each one of said pair being disposed for takeover in response to said persistent memory.

6. Apparatus as in claim **3**, wherein

each said server is disposed for transmitting a message including recovery information relating to a status of said server on recovery from a service interruption; and

each said server is disposed so that giveback of at least part of said shared resource is responsive to said recovery information.

7. Apparatus as in claim **3**, wherein

each said server is disposed for transmitting a message including recovery information relating to a status of said server on recovery from a service interruption; and

each said server is disposed so that said takeover is responsive to said recovery information.

8. Apparatus as in claim **3**, wherein

said pair includes a first server and a second server;

said first server determines a state for itself and for said second server in response to said communication path;

said second server determines a state for itself and for said first server in response to said communication path;

whereby said first server and said second server concurrently each determine state for each other, such that it does not occur that each of said first server and said second server both consider the other to be inoperative.

9. Apparatus as in claim **3**, wherein

said shared resource includes a plurality of storage devices; and

said communication path includes at least part of said storage devices.

10. Apparatus as in claim **3**, wherein

said communication path includes a plurality of independent communication paths between said pair; and

each message in said sequence includes a generation number, said generation number being responsive to a service interruption and a persistent memory for a sender of said message.

11. Apparatus as in claim **3**, wherein

said communication path includes a plurality of independent communication paths between said pair; and

said first server is disposed for determining a state for itself and for said second server in response to a state of said shared resource and in response to a state of a persistent memory at said first server.

12. Apparatus as in claim **3**, wherein

said communication path includes a plurality of independent communication paths between said pair; and

said plurality of independent communication paths includes at least two of the group: a packet network, a shared storage element, a system area network.

13. Apparatus as in claim **3**, wherein

said communication path is disposed for transmitting at least one message from a first said server to a second said server;

said message indicating that said first server is attempting said takeover;

receipt of said message being responsive to a state of said shared resource.

14. Apparatus as in claim **13**, wherein said second server is disposed for altering its state in response to said message, in said altered state refraining from writing to said shared resource.

15. A method for operating a file server, said method including steps for controlling a subset of a set of shared storage devices;

receiving and transmitting messages with a second file server, said steps for receiving and transmitting using a communication path including said shared storage devices;

monitoring said communicating path and said shared storage devices;

storing state information about said file server in a persistent memory; and

performing a takeover operation of said shared resource in response to said steps for monitoring and a state of said persistent memory.

16. A method including steps for

managing at a first server at least a part of a shared resource;

receiving and transmitting a sequence of messages between said first server to a second server, using said shared resource;

performing a takeover operation at a first server of at least part of said shared resource in response to said sequence of messages;

whereby said steps for receiving and transmitting prevent both of said first server and said second server from concurrently performing said takeover operation.

17. A method as in claim 16, including steps for

determining, at said first server, a state for itself and for said second server in response to a communication path;

determining, at said second server, a state for itself and for said first server in response to said communication path;

whereby said first server and said second server concurrently each determine state for each other, such that it does not occur that each of said first server and said second server both consider the other to be inoperative.

18. A method as in claim 16, including steps for storing state information about said first server in a persistent memory, wherein said first server determines a state for itself in response to a state of said persistent memory.

19. A method as in claim 16, including steps for

transmitting, from said first server, recovery information relating to a status of said first server on recovery from a service interruption; and

performing a giveback operation of at least part of said shared resource is responsive to said recovery information.

20. A method as in claim 16, including steps for

transmitting, from said first server, recovery information relating to a status of said server on recovery from a service interruption;

wherein said steps for performing said takeover operation are responsive to said recovery information.

21. A method as in claim 16, wherein

said shared resource includes a plurality of storage devices; and

a communication path includes at least part of said storage devices;

whereby loss of access to said part of said storage devices breaks said communication path.

22. A method as in claim 16, including steps for

transmitting at least one message from a first said server to a second said server, said message indicating that said first server is attempting said takeover;

altering a state of said second server in response to said message; and

in said altered state refraining from writing to said shared resource.

23. A method as in claim 16, wherein a communication path includes a plurality of independent communication paths between said pair; and including steps for

numbering said sequence of messages;

determining, at each recipient, a unified order for messages delivered using different ones of said plurality of independent communication paths; and

determining, at said first server, a state for itself and for said second server in response to a state of said shared resource and in response to a state of a persistent memory at said first server.

24. A method as in claim 16, wherein a communication path includes a plurality of independent communication paths between said pair; and including steps for

numbering said sequence of messages;

determining, at each recipient, a unified order for messages delivered using different ones of said plurality of independent communication paths;

transmitting substantially each message in said sequence on at least two of said plurality of independent communication paths, whereby there is no single point of failure for communication between said pair.

25. A method as in claim 16, wherein a communication path includes a plurality of independent communication paths between said pair; and including steps for

numbering said sequence of messages;

determining, at each recipient, a unified order for messages delivered using different ones of said plurality of independent communication paths;

wherein said plurality of independent communication paths includes at least two of the group: a packet network, a shared storage element, a system area network.

26. A method as in claim 16, wherein a communication path includes a plurality of independent communication paths between said pair; and including steps for

numbering said sequence of messages;

determining, at each recipient, a unified order for messages delivered using different ones of said plurality of independent communication paths;

wherein said steps for numbering include (a) determining a generation number in response to a service interruption and a persistent memory for a sender of said message, and (b) providing said generation number in substantially each message in said sequence.

* * * * *